Tesina

Seminari di Ingegneria del Software

QuOnto: Driver per SQLite e Derby e test su database di dimensione crescente

Relatore
Giuseppe De Giacomo

Studente
Francesco Menniti

INDICE

INTRODUZIONE 3
PANORAMICA SULLE ONTOLOGIE 5
STRUMENTI UTILIZZATI 7
DBMS 7
AMBIENTE DI SVILUPPO E TEST 7
STRUMENTO PER LA REALIZZAZIONE DEI GRAFICI 7
REASONER 7
CARATTERISTICHE DBMS UTILIZZATI 8
DERBY 8
SQLITE 8
DRIVER SCRITTI IN JAVA 10
DERBYDATASOURCEMANAGER.JAVA 10
SQLITEDATASOURCEMANAGER.JAVA 19
TEST EFFETTUATI 28
PREMESSE 28
ONTOLOGIE USATE NEI TEST 28
QUERY USATE NEI TEST 29
COME AVVIENE IL TEST 31
DATI E GRAFICI 31
SQLite *31*
Note sul test con SQLite *35*
Derby *36*
Note sul test con Derby *41*
CONFRONTO PRESTAZIONI QUERY TRA DERBY E SQLITE 41
Confronto query 1 ..... *41*
Confronto query 2 ..... *42*
Confronto query 3 ..... *42*
Confronto query 4 ..... *43*
Confronto query 5 ..... *43*
Confronto query 6 ..... *44*
Confronto query 7 ..... *45*
Confronto query 8 ..... *46*
Confronto query 9 ..... *47*
Confronto query 10 ..... *47*
Confronto query 11 ..... *48*
Confronto query 12 ..... *49*
Confronto query 13 ..... *49*
Confronto query 14 ..... *50*
CONFRONTO QUERY OSSERVAZIONE 51
CONCLUSIONI 52
APPENDICE A (GUIDA SQLITE) 53
USO DI SQLITE3 53
Esempio di come usare SQLite con "sqlite3.exe" *53*
UTILIZZO DELLA LIBRERIA SQLITE 55
Esempio di come Integrare il DBMS nell'applicazione Java con SQLite *55*

APPENDICE B (GUIDA DERBY).. 57

UTILIZZO DELLA LIBRERIA DERBY 57

Esempio di come integrare il DBMS nell'applicazione java con Derby.................................. 57

USO CLIENT PER DERBY: SQUIRREL SQL CLIENT 58

Introduzione

Questo lavoro aveva come obiettivo quello di realizzare e testare dei driver QuOnto ("**Qu**erying **Onto**logies" sistema sviluppato per fornire algoritmi di ragionamento automatico efficienti su Basi di Conoscenza contenenti grandi quantità di istanze: tale sistema è basato su una particolare Logica Descrittiva, DL-LiteA.) affinché possa essere in grado di comunicare con due DBMS quali SQLite (una libreria C Open Source che implementa un DBMS SQL incorporabile all'interno di applicazioni) e Derby (un piccolo database relazionale Open Source facente parte dell' "Apache DB Project"che può essere integrato in applicazioni Java).
La realizzazione di tali driver consiste nell'implementare l'interfaccia "AbstractDataSourceManager" che contiene i metodi che devono essere implementati affinché QuOnto funzioni con tali DBMS, ed estendere la classe astratta "AbstractQuontoBaseDomain" che è la classe che sta alla base di tutte le classi che implementano l'interfaccia del core di QuOnto. Prima di iniziare a scrivere codice è stato necessario reperire tutta la documentazione possibile sia su Derby sia su SQLite e tentare di capire se tali DBMS fornivano le opportune funzionalità affinché fosse possibile realizzare un driver. In generale le funzionalità dovevano riguardare la compatibilità con lo standard SQL-92 ed in particolare la possibilità di creare e leggere query. Iniziato tale lavoro di raccolta delle informazioni, è stata valutata l'effettiva possibilità di realizzare driver con successo, in quanto sia Derby sia SQLite fornivano la quasi completa compatibilità con lo standard SQL-92, e per quel che riguardava la realizzazione di opportuni driver per QuOnto non dovevano esserci problemi[1].
Successivamente è iniziata la scrittura del codice delle classi "DerbyDataSourceManager" (per Derby) e "SQLiteDataSourceManager" (per SQLite) seguendo l'esempio del driver già realizzato relativo a MySQL. Una volta realizzati i driver sono state create 4 Aboxes sia per SQLite che per Derby partendo dall'ontologia Lehigh University BenchMark (LUBM) (sviluppata a sua volta all'interno di una suite di benchmarking che potesse permettere di valutare i Semantic Web repositories o Knowledge Base Systems in modo standard e sistematico), ponendo l'accento su queries estensionali poste su grandi datasets aventi come riferimento una singola e realistica ontologia, tutto ciò per valutare la bontà di QuOnto nell'uso di questi DBMS in relazione della dimensione dei dati; infatti una Abox era relativa ad una università, una relativa a 5 università, poi 10 università ed infine 30 università. Per la realizzazione di tali Aboxes è stato utilizzato un tool scritto interamente in Java in grado di tradurre automaticamente le Aboxes scritte in formato OWL in Aboxes scritte in formato XML, secondo la DTD definita in QuOnto. Poi sono stati eseguiti i test e memorizzati tutti i risultati su file di testo e non solo, ossia i dati sono stati riportati su fogli exel e realizzati dei grafici per mostrare in modo più chiaro i risultati ottenuti.
Dopodichè è stata scritta una presentazione nella quale si riportano brevemente le caratteristiche fondamentali dei due DBMS in questione, quali la compatibilità con lo standard SQL-92, le licenze offerte, gli obiettivi di tali DBMS, i risultati sperimentali ecc. .
Infine sono state prodotte due guide, una relativa all'utilizzo di SQLite e l'altra relativa a Derby. Nella guida relativa ad SQLite vi è inizialmente una piccola introduzione nella quale viene spiegato brevemente cosa è SQLite. Poi vengono riportate le istruzioni relative a come utilizzare SQLite versione 3 per mezzo dell'interfaccia a riga di comando "sqlite3.exe" offerta dal produttore stesso di SQLite che implementa il motore di SQLite. Seguono poi le istruzioni relative all'uso della libreria SQLite ed un esempio completo di come integrare il DBMS nell'applicazione java con SQLite completa di codice ed istruzioni per la compilazione ed esecuzione con inoltre delle screenshot che illustrano con chiarezza il tutto.
Nella guida relativa a Derby vi è inizialmente, come nella guida di SQLite, una breve introduzione sul DBMS Derby. Dopodichè viene descritto in modo dettagliato come utilizzare la libreria Derby ed un esempio completo di come sia possibile integrare il DBMS nell'applicazione java con Derby;

[1] Per maggiori dettagli sulla compatibilità di SQLite vedere Appendice A.

anche in questo caso l’esempio è completo di codice, istruzioni per la compilazione ed esecuzione e screenshot dimostrative. Infine su tale guida è presente una dettagliatissima spiegazione su come utilizzare il client “SQuirrel SQL Client” (uno dei tanti utilizzabili e scaricabili dalla rete) con Derby; tale spiegazione contiene istruzioni su dove trovarlo, sull’installazione, sulla prima esecuzione, sulla creazione di un piccolo database e sull’interrogazione di quest’ultimo.

Panoramica sulle ontologie

Nell' informatica, l'ontologia viene utilizzata per formulare uno schema concettuale esaustivo e rigoroso nell'ambito di un dato dominio, in particolare definisce il tipo di entità che esistono in un determinato dominio, le relazioni esistenti fra di esse, le regole, gli assiomi ed i vincoli specifici del dominio. Riportiamo di seguito una delle definizioni accettate del termine ontologia in informatica:

> *"the systematic, formal, axiomatic development of the logic of all forms and modes of being" (Cocchiarella, 1991)*

Le ontologie sono applicate comunemente nel campo dell'intelligenza artificiale e nella rappresentazione e nella condivisione della conoscenza. Le applicazioni informatiche possono utilizzare un'ontologia per svariati scopi, fra cui il ragionamento deduttivo, la classificazione, problem solving, oltre che per facilitare la comunicazione e lo scambio di informazioni fra diversi sistemi.
Per cogliere appieno l'utilità delle ontologie si ha necessità di esprimerle in una notazione concreta che le rendi davvero utilizzabili. Un linguaggio per le ontologie è un linguaggio formale con cui viene costruita l'ontologia. Esistono diversi linguaggi in grado di definire ontologie; il W3C ha raccomandato di utilizzare per la descrizione di ontologie nel Web RDF/RDFS ed OWL. Questi linguaggi sono in grado di esprimere la conoscenza in termini di concetti e relazioni e fornire procedure di ragionamento automatico per la derivazione di conoscenza.
Questi linguaggi presentano differenti qualità in relazione all'espressività e alla complessità di ragionamento. Ad esempio, OWL è costruito sopra RDF/RDFS aggiungendo degli ulteriori costrutti per descrivere meglio le proprietà degli oggetti e delle classi, cosa che da una parte ne migliora evidentemente l'espressività, ma d'altro canto ne aumenta obbligatoriamente la complessità di ragionamento. OWL prevede tre livelli di complessità crescenti ognuno dei quali è una estensione del precedente:

> OWL Lite: supporta la classificazione gerarchica ed i vincoli semplici; presenta evidenti restrizioni in termini di espressività (ad es. consente solo cardinalità 0 o 1);
>
> OWL DL: fornisce la massima espressività pur mantenendo la completezza computazionale (tutte le conclusioni hanno garanzia di essere calcolate) e la decidibilità (tutte le computazioni terminano in un tempo finito). Presenta alcune restrizioni come ad esempio la limitazione sulla separazione del tipo (una classe non può essere un individuo o una proprietà).
>
> OWL Full: massima espressività ma senza alcuna garanzia computazionale.

Il fondamento logico di OWL DL è la logica descrittiva DL che a sua volta è un frammento decidibile di FOL (First Order Logic); ciò significa che inferire su ontologie OWL può essere fatto mediante reasoners DL.
Una base di conoscenza KB è caratterizzata da due componenti, una "TBox" (Terminological Box) ed una "ABox" (Assertion Box). L' TBox specifica le proprietà generali di concetti (o classi) e ruoli (o relazioni) ossia rende disponibile ed accessibile la rappresentazione formale del modello concettuale della porzione di realtà contenenti tutti quegli enunciati terminologici assunti come veri (livello intenzionale della conoscenza), mentre la ABox specifica le istanze di concetti e ruoli ossia rende disponibile ed accessibile la rappresentazione formale del modello concreto di una porzione di realtà, essa contiene asserzioni che specificano le conoscenze (livello estensionale della conoscenza).

Tra la semantica della ABox e la semantica delle basi di dati vi è una differenza molto importante, mentre nella prima è compatibile con una situazione di conoscenza parziale (ciò significa che tutto ciò che è contenuto nella ABox è vero mentre tutto ciò che non è contenuto non è né vero ne falso), nelle basi di dati vige l'assunzione di mondo chiuso (secondo la quale tutto ciò che è contenuto nella ABox è vero, mentre tutto ciò che non vi è contenuto è falso).
Tra le varie famiglie di logiche descrittive noi prendiamo in considerazione quella chiamata DL-Lite, una particolare logica descrittiva opportunamente dimensionata per catturare i linguaggi base di ontologie mantenendo tutti i task di ragionamento trattabili, ed in particolare, con complessità di tempo polinomiale rispetto alla dimensione della base di conoscenza (Knowledge Base KB).
Il motivo per cui ci occupiamo delle logiche descrittive appartenenti alla famiglia delle DL-Lite è perché recenti studi hanno dimostrato che nessuna variante di OWL (a meno di restrizioni particolari) sono particolarmente indicate per rappresentare e gestire le ontologie, in quanto tutte le varianti di OWL sono coNP-Hard rispetto ai dati.
La famiglia delle DL-Lite sono invece in grado di rispondere a complesse query (chiamate query congiuntive) con complessità LOG-SPACE rispetto alla dimensione dei dati; inoltre, cosa molto importante, tali logiche permettono di delegare il processamento delle query, dopo ovviamente una fase di preparazione dei dati, a DBMS.
In particolare a noi interessa la logica descrittiva DL-Lite$_A$ che è utilizzata in QuOnto. DL-Lite$_A$ TBox impone la condizione che ogni regola funzionale non può essere specializzata usando il lato destro della regola dell'asserzione di inclusione; la stessa condizione è imposta su ogni attributo funzionale (regola o concetto).

Strumenti utilizzati

Dbms

I dbms utilizzati per la realizzazione di questa tesina sono stati **Derby** e **SQLite**. Derby è un motore Open Source Database Technology parte dell' Apache DB Project, mentre SQLite è una libreria C che implementa un DBMS SQL incorporabile all'interno di applicazioni.

Ambiente di sviluppo e test

Come ambiente di sviluppo per la realizzazione dei driver è stato utilizzato **Eclipse** che fa parte di un progetto open source legato alla creazione e allo sviluppo di una piattaforma di sviluppo ideata da un consorzio di grandi società quali Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP e Serena Software, chiamato Eclipse Foundation, e creata da una comunità strutturata sullo stile dell'open source.

Strumento per la realizzazione dei grafici

Per la realizzazione delle tabelle e grafici è stato utilizzato Microsoft Excel, il foglio elettronico prodotto da Microsoft, dedicato alla produzione ed alla gestione dei fogli elettronici. È parte della suite di software di produttività personale Microsoft Office, ed è disponibile per i sistemi operativi Windows e Macintosh.

Reasoner

Questa tesina è stata realizzata per fare del testing su DBMS come SQLite e Derby su **QuOnto** "**Qu**erying **Onto**logies" sistema sviluppato per fornire algoritmi di ragionamento automatico efficienti su Basi di Conoscenza contenenti grandi quantità di istanze: tale sistema è basato su una particolare Logica Descrittiva, DL-Lite$_A$. Una delle caratteristiche che esalta QuOnto è il query answering con complessità computazionale LOGSPACE rispetto alla dimensione dei dati.

Caratteristiche DBMS utilizzati

Di seguito riporteremo obiettivi, caratteristiche, compatibilità e note dei DBMS che verranno utilizzati per la realizzazione dei driver e del successivo test mediante QuOnto.

Derby

Derby e' un piccolo database relazionale Open Source facente parte dell' "Apache DB Project" che può essere integrato in applicazioni Java. La sua principale caratteristica è la completa compatibilità con lo standard SQL-92. Derby è scritto interamente in java, è open-source e fornisce supporto JDBC e può funzionare come DB-Integrato. Per Derby esistono diverse interfacce che permettono di utilizzare le sue funzionalità come ad esempio "SQuirreL SQL Client".
Gli obiettivi del DBMS Derby sono principalmente i seguenti:

- Write One Run Everywhere
- Buona usabilità
- Conformità agli standard (SQL-92)!
- Sicurezza
- Portabilità (JAVA) !

I punti su cui Derby fonda principalmente la sua attenzione sono la conformità agli standard e la portabilità.
Per quanto riguarda la conformità, si attiene allo standard SQL-92 rispettando:

- Tables, indexes, views, triggers, procedures, functions, temporal table
- Foreing keys and constraints
- Joins, cost based optimizer
- Deadlock detection, crash recovery, backup and restore ability, Multi-user, transactions API

Mentre la portabilità è garantita dal fatto di essere scritto interamente in Java. Va inoltre sottolineato che Derby è un motore Open-Source.
Derby fornisce supporto JDBC (Java Batabase Connectivity), un connettore per database che consente l'accesso alle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato; JDBC è costituita da una API, raggruppata nel package java.sql, che serve ai client per connettersi a un database; tale supporto JDBC fornisce metodi per interrogare e modificare i dati di database relazionali.
Tale DB può essere facilmente integrato nelle applicazioni sempre grazie al supporto JDBC che lo permette.

SQLite

SQLite è una libreria C che implementa un DBMS SQL incorporabile all'interno di applicazioni. E' un software Open Source. Essendo una libreria, non è un processo standalone utilizzabile di per sé, ma può essere linkato all'interno di un altro programma. È utilizzabile con C/C++, JAVA. Esistono molti bindings per Java, tutti disponibili dal sito ufficiale. In generale esistono i bindings native e quelli 100% Java. I primi sfruttano la dll originale di SQLite, e forniscono solo un'interfaccia a tale dll tramite una libreria jar. I bindigs 100% invece hanno prestazioni leggermente minori, ma hanno il vantaggio di essere totalmente multipiattaforma, proprio perchè 100% Java.
Esistono inoltre varie interfacce che permettono di usare SQLite per la gestione di database, e dal sito ufficiale stesso è possibile scaricare l'interfaccia a riga di comando che offre la possibilità di utilizzare le funzionalità di SQLite. SQLite è in grado di gestire una basi di dati intera in un unico

file e gli accessi sono regolati dai permessi relativi al file che contiene la basi di dati, quindi i privilegi sono gestiti dal sistema operativo.
Gli obiettivi del DBMS SQLite sono principalmente:

- Buona usabilità
- Conformità agli standard
- Semplicità e leggerezza!
- Efficienza
- Portabilità

I punti di forza di SQLite sono principalmente la semplicità e la leggerezza per essere utilizzato con buoni esiti soprattutto nel web. Esistono numerosi binding e wrapper che permettono a SQLite di avere una buona portabilità. Non necessita di installazione e fondamentalmente viene utilizzato come DB integrato nelle applicazioni sia stand-alone che applicazioni in tecnologie web.
Per quanto riguarda la compatibilità con lo standard SQL-92, SQLite manca di alcune cose. Di seguito riportiamo una tabella che elenca tali mancanze:

Vincoli sulle FOREIGN KEYS	Le chiavi FOREIGN KEYS sono riconosciute, ma non implementate
Supporto completo dei trigger	Esiste un supporto per i trigger, ma non è completo. Le sotto-funzionalità mancanti sono: i trigger FOR EACH STATEMENT, quelli INSTEAD OF sulle tabelle e i trigger ricorsivi
Supporto completo ALTER TABLE	Sono supportate solo le varianti RENAME TABLE, ADD COLUMN. Mancano: DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT, etc.
Transazioni nested	Attulamente è possible una sola transazione attiva
RIGHT e FULL OUTER JOIN	Il LEFT OUTER JOIN è implementato, mancano il RIGHT e il FULL
Scrivere su VIEWs	Le view sono di sola lettura. Non è possibile eseguire i comandi INSERT, DELETE, UPDATE su di esse. E' possibile però creare un trigger sul tentativo di esecuzione di uno di questi comandi
GRANT e REVOKE	Siccome SQLite scrive e legge su un normale file, gli unici permessi d'accesso che possono essere applicati sono quelli dettati dal sistema operativo sottostante. I comandi GRANT e REVOKE non sono implementati perché sarebbero inutili

Per utilizzare SQLite esistono diverse interfacce, tra cui quella offerta dal sito produttore "SQLite3" che è una semplicissima interfaccia a riga di comando della quale verrà parlato nel capitolo relativo alla guida ad SQLite.

Driver scritti in JAVA

DerbyDataSourceManager.java

```
/**
 * @(#)DerbyDataSourceManager.java
 */

package it.uniroma1.dis.quonto.datasourcemanager.impl;

import it.uniroma1.dis.quonto.core.IEvaluationResult;
import it.uniroma1.dis.quonto.core.domain.IBasicValueSet;
import it.uniroma1.dis.quonto.core.domain.IDomainFactory;
import it.uniroma1.dis.quonto.core.domain.ISet;
import it.uniroma1.dis.quonto.core.domain.IUnionOfConjunctiveQueries;
import it.uniroma1.dis.quonto.core.domain.impl.AbstractQuontoBaseDomain;
import it.uniroma1.dis.quonto.core.domain.impl.AtomicConcept;
import it.uniroma1.dis.quonto.core.domain.impl.AtomicRole;
import it.uniroma1.dis.quonto.core.domain.impl.AtomicValueSet;
import it.uniroma1.dis.quonto.core.domain.impl.ConceptAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.ConceptAttributeDomain;
import it.uniroma1.dis.quonto.core.domain.impl.ConceptAttributeRange;
import it.uniroma1.dis.quonto.core.domain.impl.ExistentialRole;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedBasicConcept;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedBasicRole;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedBasicValueSet;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedConceptAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedRoleAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.RoleAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.RoleAttributeDomain;
import it.uniroma1.dis.quonto.core.domain.impl.RoleAttributeRange;
import it.uniroma1.dis.quonto.datasourcemanager.AbstractDataSourceManager;
import it.uniroma1.dis.quonto.datasourcemanager.exceptions.DataLevelConfigurationException;
import it.uniroma1.dis.quonto.datasourcemanager.exceptions.DataLevelSQLError;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;

public class DerbyDataSourceManager extends AbstractQuontoBaseDomain implements
AbstractDataSourceManager{

        private Connection connection = null;
        private String driver;
        private String url;
        private String username;
        private String password;

        private static final String getViewsName = "SELECT TABLE_NAME FROM information_schema.VIEWS
WHERE TABLE_SCHEMA = ?";

        private Collection<Statement> statements;

        private Collection<ResultSet> resultSets;

        private static IDomainFactory domainFactory;

        public DerbyDataSourceManager() {
        }

        public Connection getConnection() {
                return this.connection;
        }

        public DerbyDataSourceManager(IDomainFactory df, String driver, String url, String user,
String password) throws SQLException, ClassNotFoundException {
                super(df);
                DerbyDataSourceManager.domainFactory = df;
```

```
			//caricamento della classe driver
			Class.forName(driver);
			this.driver = driver;
			this.url = url;
			this.username = user;
			this.password = password;
			//apertura di una connessione sul DB, la quale deve essere usata per ogni operazione
sul DB, senza aprirne altre
			this.connection = DriverManager.getConnection(url, user, password);
		}

		public void setDomainFactory(IDomainFactory df) {
			this.domainFactory = df;
		}

		public void createConnection() throws DataLevelSQLError, ClassNotFoundException {
			try {
				//caricamento della classe driver
				Class.forName(driver);
				this.connection = DriverManager.getConnection(url, username, password);
			}
			catch(SQLException sqlexeption) {
				throw new DataLevelSQLError("Data Source SQL Exception when establishing
Database connection: " + sqlexeption.getMessage());
			}
		}

public String getJDBCDriver() {
			return this.driver;
		}

		public String getJDBCUrl() {
			return this.url;
		}

		public String getJDBCUsername() {
			return this.username;
		}

		public String getJDBCPassword() {
			return this.password;
		}

		public ResultSet executeQuery(String sql) throws DataLevelConfigurationException,
DataLevelSQLError {
			if (this.connection != null) {
				try {
					Statement stmt = this.connection.createStatement();
					//ResultSet è l'insieme contenete i risultati di JDBC
					ResultSet rs = stmt.executeQuery(sql);
					return rs;
				}
				catch(SQLException sqlex) {
					sqlex.printStackTrace();
					throw new DataLevelSQLError("Data Level SQL Error executing query <" +
sql + ">: " + sqlex.getMessage());
				}
			}
			throw new DataLevelConfigurationException("Data Level Configuration Error executing
query <" + sql + ">: The JDBC connection to the data level is not available!");
		}

		public int executeUpdate(String sql) throws DataLevelSQLError,
DataLevelConfigurationException {
			if (this.connection != null) {
				try {
					Statement stmt = this.connection.createStatement();
					int res = stmt.executeUpdate(sql);
					stmt.close();
					return res;
				}
				catch(SQLException sqlex) {
					sqlex.printStackTrace();
					throw new DataLevelSQLError("Data Level SQL Error executing update
query <" + sql + ">: " + sqlex.getMessage());
				}
			}
```

```
            throw new DataLevelConfigurationException("Data Level Configuration Error executing
update query <" + sql + ">: The JDBC connection to the data level is not available!");
        }

        //sostituisce in str il singolo apice con il doppio apice, serve nel caso in cui
l'inserimento di un campo comporti che ci
        //sia un apice, in quanto     nella sintassi di DERBY, un inserimento di un valore com apice
in un campo, si fa con un doppio apice,
        //ES:INSERT INTO ANAGRAFICA VALUES (14,'LUCA', 'D''ARPINO', '06/02/1982')
        public String escapeQuotes(String str) {
                return str.replace("'","''");
        }

        //produce e ritorna una stringa che rappresenta l'espressione usata per gli inserimenti nel
DB
        public String getInsertStatement(String table, String[] cols, String[] values, boolean[]
quote) throws DataLevelSQLError {
                if ((values == null) || (values.length == 0))
                        throw new DataLevelSQLError("Cannot create insert statement for table "
                                        + table + ": the values array cannot be empty!");
                if ((quote == null) || (quote.length == 0))
                        throw new DataLevelSQLError("Cannot create insert statement for table "
                                        + table + ": the quotation array cannot be empty!");
                if (cols != null) {
                        if (cols.length == 0)
                                throw new DataLevelSQLError("Cannot create insert statement for table
"
                                                + table + ": columns number cannot be empty! (use null
instead)");
                        if (cols.length != values.length)
                                throw new DataLevelSQLError("Cannot create insert statement for table
"
                                                + table + ": columns number does not fit with values
number!");
                }
                //se il numero di indici di colonna è diverso dal numero di valori da inserire lancia
eccezione
                if (values.length != quote.length)
                        throw new DataLevelSQLError("Cannot create insert statement for table "
                                        + table + ": values number does not fit with quotes
information!");
                //creo la stringa per l'inserimento di records
                String insert = "INSERT INTO " + table;
                //indico i nomi delle colonne
                if (cols != null) {
                        insert += "(";
                        insert += cols[0];
                        for (int i=1; i < cols.length; i++)
                                insert += "," + cols[i];
                        insert += ")";
                }
                insert += " VALUES (";
                insert += (quote[0]?"'":"") + (quote[0]?this.escapeQuotes(values[0]):values[0]) +
(quote[0]?"'":"");
                for (int i=1; i < values.length; i++)
                        insert += ", " + (quote[i]?"'":"") +
(quote[i]?this.escapeQuotes(values[i]):values[i]) + (quote[i]?"'":"");
                insert += ");";
                return insert;
        }
        //Ritorna un result set vuoto
        public ResultSet getEmptyResultSet(IUnionOfConjunctiveQueries ucq) throws DataLevelSQLError {
                try {
                        Statement stmt = this.connection.createStatement();
                        ResultSet rs = stmt.executeQuery("select * from mysql.user where 0 = 1;");
                        return rs;
                }
                catch(SQLException sqlex) {
                        sqlex.printStackTrace();
                        throw new DataLevelSQLError("Data Level SQL Error evaluating query <" + ucq +
">: " + sqlex.getMessage());
                }
        }
        //esegue le query prese come agomento
        public ResultSet runQuery(IUnionOfConjunctiveQueries ucq) throws
DataLevelConfigurationException, DataLevelSQLError {
                if (this.connection != null) {
                        if (ucq.getConjunctiveQueries().isEmpty()) return getEmptyResultSet(ucq);
```

```
			try {
				Statement stmt = this.connection.createStatement();
				ResultSet rs = stmt.executeQuery(ucq.getDirectMappedSqlQuery());
				return rs;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level SQL Error evaluating query <"
+ ucq + ">: " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level Configuration Error evaluating
query <" + ucq + ">: The JDBC connection to the data level is not available!");
	}
//Returns a collection of string representing the name of the columns of the result set of the
//given <code>sql</code> query, by invoking the suitable methods provided through the
//<code>ResultSetMetaData</code> interface.
	public Collection<String> getSQLColumnNames(String sql) throws
DataLevelConfigurationException, DataLevelSQLError {
		if (this.connection != null) {
			try {
				Collection<String> names = new HashSet<String>();
				PreparedStatement stmt = connection.prepareStatement(sql);
				ResultSetMetaData rsmd = stmt.getMetaData();
				int cols = rsmd.getColumnCount();
				for (int i=0; i < cols; i++)
					names.add(rsmd.getColumnName(i+1));
				stmt.close();
				return names;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level SQL Error in getting column
names for query <" + sql + ">: " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level SQL Error in getting column
names for query <" + sql + ">: The JDBC connection to the data level is not available!");
	}
	/**
	 * Drops the view named <code>name</code> from the underlying database connection
	 *
	 * @param name the name of the view being dropped
	 * @return the result of the update statement
	 * @throws DataLevelConfigurationException thrown when a database connection is not available
or has not been instantiated
	 * @throws DataLevelSQLError if some errors are raised from the underlying JDBC calls
	 *
	 *N.B. In Derby il comando per rimuovere una vista è "DROP VIEW + nomeVista"
	 *
	 */
	public int dropView(String name) throws DataLevelConfigurationException, DataLevelSQLError {
		if (this.connection != null) {
			try {
				//imporante lavorare sempre sulla stessa connessione
				Statement stmt = this.connection.createStatement();
				String drop = "DROP VIEW " + name;
				int res = stmt.executeUpdate(drop);
				stmt.close();
				return res;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				//N.B. DROP VIEW is disallowed if there are any views or open cursors
dependent on the view
				throw new DataLevelSQLError("Data Level SQL Error in dropping view <"
+ name + "> : N.B. DROP VIEW is disallowed if there are any views or open cursors dependent on the
view:  " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level Configuration Error in dropping
view <" + name + "> : The connection to the data level is not available!");
	}

	/**
	 * Creates a new view named <code>name</code>, with the specified <code>body</code>, over the
underlying database connection
	 *
```

```
	 * @param name the name of the view
	 * @param body the body of the view
	 * @return the result of the update command
	 * @throws DataLevelConfigurationException  thrown when a database connection is not
available or has not been instantiated
	 * @throws DataLevelSQLError if some errors are raised from the underlying jdbc calls
	 */
	public int createView(String name, String body) throws DataLevelConfigurationException,
DataLevelSQLError {
		if (this.connection != null) {
			try {
				//imporante lavorare sempre sulla stessa connessione
				Statement stmt = this.connection.createStatement();
				String create = "CREATE VIEW " + name + " AS " + body;
				int res = stmt.executeUpdate(create);
				stmt.close();
				return res;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level Configuration Error in
creating view <" + name + "> : " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level Configuration Error in creating
view <" + name + "> :The connection to the data level is not available!");
	}

	/**
	 * Creates a SQL column statement that produces a logic term by using suitable SQL statement
for string manipulation. Example,
	 * given the <code>functor</code> <code>f</code> and the array of <code>sqlVariables</code>
<code>v1, v2, ... , vn</code>, the output
	 * of the SQL statement produced by this method should look like <code>f(v1,v2, ... ,
vn)</code>
	 *
	 * @param functor a string representing a function symbol
	 * @param sqlVariables an array of string representing SQL variables
	 * @return
	 */
	//In Derby The concatenation operator is "||",that concatenates its right operand to the end
of its left operand
	// esempio: f(a,b,c,d)  -->  'f' || '(' || 'a' || ',' || 'b' || ',' || 'c' || ',' || 'd' ||
')'
	public String createFunctorConcatStatement(String functor, String[] sqlVariables) {
		String str = "'" + functor + "' || '(' || '";
		str += sqlVariables[0];
		for (int i = 1; i < sqlVariables.length; i++)
			str += "' || ',' || '" + sqlVariables[i];
		return str.concat("' || ')'");
	}

	/**
	 * Given a collection of string <code>bodies</code> representing arbitrary SQL
	 * queries returning result set of the same arity,produces a SQL union statement
	 * according to the syntax of the underlying database management system
	 *
	 * @param bodies a set of arbitrary SQL statements
	 * @return a SQL union of conjunctive queries
	 *
	 *     UNION builds an intermediate ResultSet with all of the rows from both queries
	 *  and eliminates the duplicate rows before returning the remaining rows. UNION ALL
	 *  returns all rows from both queries as the result.
	 *
	 *N.B. In Derby il DISTINCT è di default
	 *
	 *ESEMPIO: (SELECT ID, COGNOME
				FROM ANAGRAFICA
				WHERE ID>4)
				UNION
				(SELECT ID, COGNOME
				FROM ANAGRAFICA
				WHERE ID>1)
	 */

	public String createUnionDistinct(Set<String> bodies) {
		if (bodies.isEmpty()) return null;
		StringBuffer buffer = new StringBuffer();
```

```
		for (String body : bodies) {
			if (body.charAt(body.length()-1) == ';')
				body = body.substring(0, body.length() -1);
			body = "(".concat(body.concat(")"));
			buffer.append(body + " UNION ");
		}
		String ret = buffer.toString();
		return ret.substring(0, ret.length() - " UNION ".length());
	}

	/**
	 * Creates an SQL statement for the creation of a view with the header
<code>viewHeader</code> and the body <code>
	 *
	 * @param viewHeader the header of the view
	 * @param viewBody the body of the view
	 * @return
	 */
	public String getCreateViewStatement(String viewHeader,	String viewBody) {
		String create = "CREATE VIEW " + viewHeader + " AS " + viewBody;
		return create;
	}

	public void setJDBCDriver(String driver) throws DataLevelConfigurationException {
		if ((driver == null) || (driver.equals("")))
				throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		this.driver = driver;
	}

	public void setJDBCPassword(String password) throws DataLevelConfigurationException {
		if ((password == null) || (password.equals("")))
			this.password="APP";
	}

	public void setJDBCUrl(String url) throws DataLevelConfigurationException {
		if ((url == null) || (url.equals("")))
				throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		this.url = url;
	}

	public void setJDBCUsername(String username) throws DataLevelConfigurationException {
		if ((username == null) || (username.equals("")))
			this.username ="APP";
	}

	public String toString() {
		String desc = " --- Data Source Manager ---";
		desc += "\nClass:     " + this.getClass().getName();
		desc += "\nDriver:    " + this.getJDBCDriver();
		desc += "\nUrl:       " + this.getJDBCUrl();
		desc += "\nUsername:  " + this.getJDBCUsername();
		desc += "\nPassword:  " + this.getJDBCPassword();
		return desc;
	}

	public void createDatabase(String string) throws SQLException,
DataLevelConfigurationException {
		throw new UnsupportedOperationException("Cannot create Database");
	}
////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////
	public void closeConnection() throws DataLevelSQLError {
		try {
			this.closePendingJDBCObjects();
			this.connection.close();
		}
		catch(SQLException ex) {
			throw new DataLevelSQLError("Cannot close the connection: " +
ex.getMessage());
		}

	}

	public void closePendingJDBCObjects() throws DataLevelSQLError {
		for (ResultSet rs: this.resultSets) {
			try {
```

```
					rs.close();
				} catch (SQLException e) {
					throw new DataLevelSQLError("Cannot close JDBC result set: " +
e.getMessage());
				}
			}
			for (Statement stmt : this.statements) {
				try {
					stmt.close();
				} catch (SQLException e) {
					throw new DataLevelSQLError("Cannot close JDBC statement: " +
e.getMessage());
				}
			}
	}

	public void destroyDatabase(String dbName) throws SQLException,
				DataLevelConfigurationException {
			throw new UnsupportedOperationException("Cannot destroy Database");
	}

	public String getDeleteAllFromTableStatement(String tableName) {
			String delete = "DELETE FROM " + tableName;
			return delete;
	}

	public String getDirectMappedSetTableCreationStatement(ISet gen,
				HashMap<IBasicValueSet, String> valueSetTypes) {
			String sqlStr = "CREATE TABLE ";
			String sqlObjectType = this.getTypeConversionTable().get("Object");

			if(gen instanceof AtomicConcept) {

				AtomicConcept atomicConc = (AtomicConcept)gen;
				sqlStr += atomicConc.getName() + " (term " + sqlObjectType + ", PRIMARY
KEY(term));";
			}

			else if(gen instanceof ExistentialRole) {

				ExistentialRole exRole = (ExistentialRole)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(exRole.getBasicRole(),
valueSetTypes);
			}

			else if(gen instanceof ConceptAttributeDomain) {

				ConceptAttributeDomain CADom = (ConceptAttributeDomain)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(CADom.getConceptAttribute(),
valueSetTypes);
			}

			else if (gen instanceof NegatedBasicConcept) {

				NegatedBasicConcept negConc = (NegatedBasicConcept)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(negConc.getNegatedConcept(),
valueSetTypes);
			}

			else if(gen instanceof AtomicRole) {

				AtomicRole atomicRole = (AtomicRole)gen;
				sqlStr += atomicRole.getName() +
				" (term1 " + sqlObjectType + ", term2 " + sqlObjectType + ", " +
				"PRIMARY KEY(term1, term2));";
			}

			else if(gen instanceof RoleAttributeDomain) {

				RoleAttributeDomain RADom = (RoleAttributeDomain)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(RADom.getRoleAttribute(),
valueSetTypes);
			}

			else if(gen instanceof NegatedBasicRole) {

				NegatedBasicRole negRole = (NegatedBasicRole)gen;
```

```
                        sqlStr = getDirectMappedSetTableCreationStatement(negRole.getNegatedRole(),
valueSetTypes);
                }

                else if(gen instanceof AtomicValueSet) {

                        AtomicValueSet atomicV = (AtomicValueSet)gen;
                        String sqlVType = valueSetTypes.get(atomicV);
                        sqlStr += atomicV.getName() + " (term " + sqlVType + ", PRIMARY KEY(term));";
                }

                else if(gen instanceof ConceptAttributeRange) {

                        ConceptAttributeRange CARange = (ConceptAttributeRange)gen;
                        sqlStr =
getDirectMappedSetTableCreationStatement(CARange.getConceptAttribute(), valueSetTypes);
                }

                else if(gen instanceof RoleAttributeRange) {

                        RoleAttributeRange RARange = (RoleAttributeRange)gen;
                        sqlStr = getDirectMappedSetTableCreationStatement(RARange.getRoleAttribute(),
valueSetTypes);
                }

                else if(gen instanceof NegatedBasicValueSet) {

                        NegatedBasicValueSet negV = (NegatedBasicValueSet)gen;
                        sqlStr = getDirectMappedSetTableCreationStatement(negV.getNegatedValueSet(),
valueSetTypes);

                }

                else if(gen instanceof ConceptAttribute) {

                        ConceptAttribute concAttr = (ConceptAttribute)gen;
                        String sqlVType =
valueSetTypes.get(this.getDomainFactory().getConceptAttributeRange(concAttr));
                        sqlStr += concAttr.getName() +
                        " (term1 " + sqlObjectType + ", term2 " + sqlVType + ", " +
                        "PRIMARY KEY(term1, term2));";
                }

                else if(gen instanceof NegatedConceptAttribute) {

                        NegatedConceptAttribute negConcAttr = (NegatedConceptAttribute)gen;
                        sqlStr =
getDirectMappedSetTableCreationStatement(negConcAttr.getNegatedConceptAttribute(), valueSetTypes);
                }

                else if(gen instanceof RoleAttribute) {

                        RoleAttribute roleAttr = (RoleAttribute)gen;
                        String sqlVType =
valueSetTypes.get(this.getDomainFactory().getRoleAttributeRange(roleAttr));
                        sqlStr += roleAttr.getName() +
                        " (term1 " + sqlObjectType + ", term2 " + sqlObjectType + ", term3 " +
sqlVType + ", " +
                        "PRIMARY KEY(term1, term2, term3));";
                }

                else {                          // NegatedRoleAttribute

                        NegatedRoleAttribute negRoleAttr = (NegatedRoleAttribute)gen;
                        sqlStr =
getDirectMappedSetTableCreationStatement(negRoleAttr.getNegatedRoleAttribute(), valueSetTypes);
                }

                return sqlStr;
        }

        public String getEmptyTable(int arity) {
                String attributes = "";
                for (int i=0;i<arity;i++){
                        attributes+="table_name";
                        if (i!=arity-1) attributes+=",";
                }
                return "SELECT "+attributes+ " FROM information_schema.tables WHERE 0=1";
```

```
	}

	public String getEmptyTable(String[] aliasTerms) {
		int arity = aliasTerms.length;
		String attributes = "";
		for (int i=0;i<arity;i++){
			attributes+="table_name" + " AS " + aliasTerms[i];
			if (i!=arity-1) attributes+=",";
		}
		return "SELECT "+attributes+ " FROM information_schema.tables WHERE 0=1";

	}

	public Set<String> getListOfActiveViews() throws SQLException,
			DataLevelConfigurationException, DataLevelSQLError {
		if (this.connection!=null) {
			Set<String> setOfActiveViews = new HashSet<String>();
			PreparedStatement getViewsName =
this.getConnection().prepareStatement(this.getViewsName);
			getViewsName.setString(1, this.getSchemaDbName());
			ResultSet rs = getViewsName.executeQuery();
			while(rs.next()) {
				String viewName = rs.getString(1).toLowerCase();
				setOfActiveViews.add(viewName);
			}
			return setOfActiveViews;
		}
		else return null;
	}

	public String getSchemaDbName() throws SQLException {
		if (this.connection!=null) return this.connection.getCatalog();
		else return null;
	}

	public HashMap<String, String> getTypeConversionTable() {
		HashMap<String, String> map = new HashMap<String, String>();
		map.put("Object", "VARCHAR(255)");
		map.put("TopD", "VARCHAR(255)");
		map.put("xs:int", "INTEGER");
		map.put("xs:short", "SMALLINT");
		map.put("xs:string", "VARCHAR(255)");
		map.put("xs:date", "DATE");
		return map;
	}

	public void setJDBCUrl(String url_no_db, String dbname) throws
DataLevelConfigurationException {
		if ((url_no_db == null) || (url_no_db.equals("")) || (dbname == null) ||
(dbname.equals("")))
				throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		if (url_no_db.endsWith("/")) {
			this.url = url_no_db + dbname;
		}
		else {
			this.url = url_no_db + "/" + dbname;
		}
	}

}
```

SQLiteDataSourceManager.java

```java
/**
 * @(#)SQLiteDataSourceManager.java
 */

package it.uniroma1.dis.quonto.datasourcemanager.impl;

import it.uniroma1.dis.quonto.core.IEvaluationResult;
import it.uniroma1.dis.quonto.core.domain.IBasicValueSet;
import it.uniroma1.dis.quonto.core.domain.IDomainFactory;
import it.uniroma1.dis.quonto.core.domain.ISet;
import it.uniroma1.dis.quonto.core.domain.IUnionOfConjunctiveQueries;
import it.uniroma1.dis.quonto.core.domain.impl.AbstractQuontoBaseDomain;
import it.uniroma1.dis.quonto.core.domain.impl.AtomicConcept;
import it.uniroma1.dis.quonto.core.domain.impl.AtomicRole;
import it.uniroma1.dis.quonto.core.domain.impl.AtomicValueSet;
import it.uniroma1.dis.quonto.core.domain.impl.ConceptAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.ConceptAttributeDomain;
import it.uniroma1.dis.quonto.core.domain.impl.ConceptAttributeRange;
import it.uniroma1.dis.quonto.core.domain.impl.ExistentialRole;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedBasicConcept;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedBasicRole;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedBasicValueSet;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedConceptAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.NegatedRoleAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.RoleAttribute;
import it.uniroma1.dis.quonto.core.domain.impl.RoleAttributeDomain;
import it.uniroma1.dis.quonto.core.domain.impl.RoleAttributeRange;
import it.uniroma1.dis.quonto.datasourcemanager.AbstractDataSourceManager;
import it.uniroma1.dis.quonto.datasourcemanager.exceptions.DataLevelConfigurationException;
import it.uniroma1.dis.quonto.datasourcemanager.exceptions.DataLevelSQLError;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;

public class SQLiteDataSourceManager extends AbstractQuontoBaseDomain implements
AbstractDataSourceManager{

        private Connection connection = null;
        private String driver;
        private String url;
        private String username;
        private String password;

        private static final String getViewsName = "SELECT TABLE_NAME FROM information_schema.VIEWS
WHERE TABLE_SCHEMA = ?";

        private Collection<Statement> statements;

        private Collection<ResultSet> resultSets;

        private static IDomainFactory domainFactory;

    public SQLiteDataSourceManager() {
    }

    public Connection getConnection() {
                return this.connection;
        }

        public SQLiteDataSourceManager(IDomainFactory df, String driver, String url, String user,
String password) throws SQLException, ClassNotFoundException {
                super(df);
                SQLiteDataSourceManager.domainFactory = df;
                //caricamento della classe driver
                Class.forName(driver);
                this.driver = driver;
```

```
                this.url = url;
                this.username = user;
                this.password = password;
                //apertura di una connessione sul DB, la quale deve essere usata per ogni operazione
sul DB, senza aprirne altre
                this.connection = DriverManager.getConnection(url, user, password);
        }

        public void setDomainFactory(IDomainFactory df) {
                this.domainFactory = df;
        }

        public void createConnection() throws DataLevelSQLError, ClassNotFoundException {
                try {
                        //caricamento della classe driver
                        Class.forName(driver);
                        this.connection = DriverManager.getConnection(url, username, password);
                }
                catch(SQLException sqlex) {
                        throw new DataLevelSQLError("Data Source SQL Error when establishing Database
connection: " + sqlex.getMessage());
                }
        }

    public String getJDBCDriver() {
                return this.driver;
        }

        public String getJDBCUrl() {
                return this.url;
        }

        public String getJDBCUsername() {
                return this.username;
        }

        public String getJDBCPassword() {
                return this.password;
        }
//sostituisce in str il singolo apice con il doppio apice, serve nel caso in cui l'inserimento di un
campo comporti che ci
//sia un apice, in quanto     nella sintassi di DERBY, un inserimento di un valore com apice in un
campo, si fa con un doppio apice,
//ES:INSERT INTO ANAGRAFICA VALUES (14,'LUCA', 'D''ARPINO', '06/02/1982')
        public String escapeQuotes(String str) {
                return str.replace("'","''");
        }

        public ResultSet executeQuery(String sql) throws DataLevelConfigurationException,
DataLevelSQLError {
                if (this.connection != null) {
                        try {
                                Statement stmt = this.connection.createStatement();
                                ResultSet rs = stmt.executeQuery(sql);
                                return rs;
                        }
                        catch(SQLException sqlex) {
                                sqlex.printStackTrace();
                                throw new DataLevelSQLError("Data Level SQL Error executing query <" +
sql + ">: " + sqlex.getMessage());
                        }
                }
                throw new DataLevelConfigurationException("Data Level Configuration Error executing
query <"
                                + sql + ">: The JDBC connection to the data level is not available!");
        }

        public int executeUpdate(String sql) throws DataLevelSQLError,
DataLevelConfigurationException {
                if (this.connection != null) {
                        try {
                                Statement stmt = this.connection.createStatement();
                                int res = stmt.executeUpdate(sql);
                                stmt.close();
                                return res;
                        }
                        catch(SQLException sqlex) {
                                sqlex.printStackTrace();
```

```
					throw new DataLevelSQLError("Data Level SQL Error executing update
query <" + sql + ">: " + sqlex.getMessage());
				}
			}
			throw new DataLevelConfigurationException("Data Level Configuration Error executing
update query <" + sql + ">: The JDBC connection to the data level is not available!");
		}
		//produce e ritorna una stringa che rappresenta l'espressione usata per gli inserimenti nel
DB
		public String getInsertStatement(String table, String[] cols, String[] values, boolean[]
quote) throws DataLevelSQLError {
			if ((values == null) || (values.length == 0))
				throw new DataLevelSQLError("Cannot create insert statement for table "
						+ table + ": the values array cannot be empty!");
			if ((quote == null) || (quote.length == 0))
				throw new DataLevelSQLError("Cannot create insert statement for table "
						+ table + ": the quotation array cannot be empty!");
			if (cols != null) {
				if (cols.length == 0)
					throw new DataLevelSQLError("Cannot create insert statement for table
"
							+ table + ": columns number cannot be empty! (use null
instead)");
				if (cols.length != values.length)
					throw new DataLevelSQLError("Cannot create insert statement for table
"
							+ table + ": columns number does not fit with values
number!");
			}
			//se il numero di indici di colonna è diverso dal numero di valori da inserire lancia
eccezione
			if (values.length != quote.length)
				throw new DataLevelSQLError("Cannot create insert statement for table "
						+ table + ": values number does not fit with quotes
information!");
			//creo la stringa per l'inserimento di records
			String insert = "INSERT INTO " + table;
			if (cols != null) {
				insert += "(";
				insert += cols[0];
				for (int i=1; i < cols.length; i++)
					insert += ", " + cols[i];
				insert += ")";
			}
			insert += " VALUES (";
			insert += (quote[0]?"'":"") + (quote[0]?this.escapeQuotes(values[0]):values[0]) +
(quote[0]?"'":"");
			for (int i=1; i < values.length; i++)
				insert += ", " + (quote[i]?"'":"") +
(quote[i]?this.escapeQuotes(values[i]):values[i]) + (quote[i]?"'":"");
			insert += ");";
			return insert;
		}
		//Ritorna un result set vuoto
		public ResultSet getEmptyResultSet(IUnionOfConjunctiveQueries ucq) throws DataLevelSQLError {
			try {
				Statement stmt = this.connection.createStatement();
				ResultSet rs = stmt.executeQuery("select * from mysql.user where 0 = 1;");
				return rs;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level SQL Error evaluating query <" + ucq +
">: " + sqlex.getMessage());
			}
		}
		//esegue le query prese come agomento
		public ResultSet runQuery(IUnionOfConjunctiveQueries ucq) throws
DataLevelConfigurationException, DataLevelSQLError {
			if (this.connection != null) {
				if (ucq.getConjunctiveQueries().isEmpty()) return getEmptyResultSet(ucq);
				try {
					Statement stmt = this.connection.createStatement();
					ResultSet rs = stmt.executeQuery(ucq.getDirectMappedSqlQuery());
					return rs;
				}
				catch(SQLException sqlex) {
					sqlex.printStackTrace();
```

```
				throw new DataLevelSQLError("Data Level SQL Error evaluating query <"
+ ucq + ">: " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level Configuration Error evaluating
query <"
				+ ucq + ">: The JDBC connection to the data level is not available!");
	}

	public Collection<String> getSQLColumnNames(String sql) throws
DataLevelConfigurationException, DataLevelSQLError {
		if (this.connection != null) {
			try {
				Collection<String> names = new HashSet<String>();
				PreparedStatement stmt = connection.prepareStatement(sql);
				ResultSetMetaData rsmd = stmt.getMetaData();
				int cols = rsmd.getColumnCount();
				for (int i=0; i < cols; i++)
					names.add(rsmd.getColumnName(i+1));
				stmt.close();
				return names;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level SQL Error in getting column
names for query <" + sql + ">: " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level SQL Error in getting column
names for query <"
				+ sql + ">: The JDBC connection to the data level is not available!");
	}

		/**
	 * Drops the view named <code>name</code> from the underlying database connection
	 *
	 * @param name the name of the view being dropped
	 * @return the result of the update statement
	 * @throws DataLevelConfigurationException thrown when a database connection is not available
or has not been instantiated
	 * @throws DataLevelSQLError if some errors are raised from the underlying JDBC calls
	 *
	 *N.B. In SQLite il comando per rimuovere una vista è "DROP VIEW IF EXIST + nomeVista"
	 *
	 */
	public int dropView(String name) throws DataLevelConfigurationException, DataLevelSQLError {
		if (this.connection != null) {
			try {
				//usare sempre la stessa connessione
				Statement stmt = this.connection.createStatement();
				String drop = "DROP VIEW IF EXISTS " + name;
				int res = stmt.executeUpdate(drop);
				stmt.close();
				return res;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level SQL Error in dropping view <"
+ name + "> : " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level Configuration Error in dropping
view <"
				+ name + "> : The connection to the data level is not available!");
	}

	/**
	 * Creates a new view named <code>name</code>, with the specified <code>body</code>, over the
underlying database connection
	 *
	 * @param name the name of the view
	 * @param body the body of the view
	 * @return the result of the update command
	 * @throws DataLevelConfigurationException  thrown when a database connection is not
available or has not been instantiated
	 * @throws DataLevelSQLError if some errors are raised from the underlying jdbc calls
	 */
```

```
	public int createView(String name, String body) throws DataLevelConfigurationException,
DataLevelSQLError {
		if (this.connection != null) {
			try {
				Statement stmt = this.connection.createStatement();
				String create = "CREATE VIEW " + name + " AS " + body;
				int res = stmt.executeUpdate(create);
				stmt.close();
				return res;
			}
			catch(SQLException sqlex) {
				sqlex.printStackTrace();
				throw new DataLevelSQLError("Data Level Configuration Error in
creating view <" + name + "> : " + sqlex.getMessage());
			}
		}
		throw new DataLevelConfigurationException("Data Level Configuration Error in creating
view <"
				+ name + "> :The connection to the data level is not available!");
	}

/**
	 * Creates a SQL column statement that produces a logic term by using suitable SQL statement
for string manipulation. Example,
	 * given the <code>functor</code> <code>f</code> and the array of <code>sqlVariables</code>
<code>v1, v2, ... , vn</code>, the output
	 * of the SQL statement produced by this method should look like <code>f(v1,v2, ... ,
vn)</code>
	 *
	 * @param functor a string representing a function symbol
	 * @param sqlVariables an array of string representing SQL variables
	 * @return
	 */
	//In SQLite The concatenation operator is "||",that concatenates its right operand to the end
of its left operand
	// esempio: f(a,b,c,d)  -->  'f' || '(' || 'a' || ',' || 'b' || ',' || 'c' || ',' || 'd' ||
')'
	public String createFunctorConcatStatement(String functor, String[] sqlVariables) {
		String str = "'" + functor + "' || '(' || '";
		str += sqlVariables[0];
		for (int i = 1; i < sqlVariables.length; i++)
			str += "' || ',' || '" + sqlVariables[i];
		return str.concat("' || ')'");
	}
	//N.B. In SQLite il DISTINCT non è di default
	public String createUnionDistinct(Set<String> bodies) {
		if (bodies.isEmpty()) return null;
		StringBuffer buffer = new StringBuffer();
		for (String body : bodies) {
			if (body.charAt(body.length()-1) == ';')
				body = body.substring(0, body.length() -1);
			body = "(".concat(body.concat(")"));
			buffer.append(body + " UNION DISTINCT ");

		}
		String ret = buffer.toString();
		return ret.substring(0, ret.length() - " UNION DISTINCT ".length());
	}

	public String getCreateViewStatement(String viewHeader,String viewBody) {
		String create = "CREATE VIEW " + viewHeader + " AS " + viewBody;
		return create;
	}

	public void setJDBCDriver(String driver) throws DataLevelConfigurationException {
		if ((driver == null) || (driver.equals("")))
				throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		this.driver = driver;
	}

	public void setJDBCPassword(String password) throws DataLevelConfigurationException {
		if ((password == null) || (password.equals("")))
				throw new DataLevelConfigurationException("JDBC password cannot be
empty!");
		this.password = password;
	}
```

```
	public void setJDBCUrl(String url) throws DataLevelConfigurationException {
		if ((url == null) || (url.equals("")))
			throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		this.url = url;
	}

	public void setJDBCUsername(String username) throws DataLevelConfigurationException {
		if ((username == null) || (username.equals("")))
			throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		this.username = username;
	}

	public String toString() {
		String desc = " --- Data Source Manager ---";
		desc += "\nClass:    " + this.getClass().getName();
		desc += "\nDriver:   " + this.getJDBCDriver();
		desc += "\nUrl:      " + this.getJDBCUrl();
		desc += "\nUsername: " + this.getJDBCUsername();
		desc += "\nPassword: " + this.getJDBCPassword();
		return desc;
	}

	public void createDatabase(String string) throws SQLException,
DataLevelConfigurationException {
		throw new UnsupportedOperationException("Cannot create Database");
	}
//////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////
	public void closeConnection() throws DataLevelSQLError {
		try {
			this.closePendingJDBCObjects();
			this.connection.close();
		}
		catch(SQLException ex) {
			throw new DataLevelSQLError("Cannot close the connection: " +
ex.getMessage());
		}

	}

	public void closePendingJDBCObjects() throws DataLevelSQLError {
		for (ResultSet rs: this.resultSets) {
			try {
				rs.close();
			} catch (SQLException e) {
				throw new DataLevelSQLError("Cannot close JDBC result set: " +
e.getMessage());
			}
		}
		for (Statement stmt : this.statements) {
			try {
				stmt.close();
			} catch (SQLException e) {
				throw new DataLevelSQLError("Cannot close JDBC statement: " +
e.getMessage());
			}
		}
	}

	public void destroyDatabase(String dbName) throws SQLException,
			DataLevelConfigurationException {
		throw new UnsupportedOperationException("Cannot use destroy database");
	}

	public String getDeleteAllFromTableStatement(String tableName) {
		String delete = "DELETE FROM " + tableName;
		return delete;
	}

	public String getDirectMappedSetTableCreationStatement(ISet gen,
			HashMap<IBasicValueSet, String> valueSetTypes) {
		String sqlStr = "CREATE TABLE ";
		String sqlObjectType = this.getTypeConversionTable().get("Object");

		if(gen instanceof AtomicConcept) {
```

```
				AtomicConcept atomicConc = (AtomicConcept)gen;
				sqlStr += atomicConc.getName() + " (term " + sqlObjectType + ", PRIMARY
KEY(term));";
			}

			else if(gen instanceof ExistentialRole) {

				ExistentialRole exRole = (ExistentialRole)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(exRole.getBasicRole(),
valueSetTypes);
			}

			else if(gen instanceof ConceptAttributeDomain) {

				ConceptAttributeDomain CADom = (ConceptAttributeDomain)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(CADom.getConceptAttribute(),
valueSetTypes);
			}

			else if (gen instanceof NegatedBasicConcept) {

				NegatedBasicConcept negConc = (NegatedBasicConcept)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(negConc.getNegatedConcept(),
valueSetTypes);
			}

			else if(gen instanceof AtomicRole) {

				AtomicRole atomicRole = (AtomicRole)gen;
				sqlStr += atomicRole.getName() +
				" (term1 " + sqlObjectType + ", term2 " + sqlObjectType + ", " +
				"PRIMARY KEY(term1, term2));";
			}

			else if(gen instanceof RoleAttributeDomain) {

				RoleAttributeDomain RADom = (RoleAttributeDomain)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(RADom.getRoleAttribute(),
valueSetTypes);
			}

			else if(gen instanceof NegatedBasicRole) {

				NegatedBasicRole negRole = (NegatedBasicRole)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(negRole.getNegatedRole(),
valueSetTypes);
			}

			else if(gen instanceof AtomicValueSet) {

				AtomicValueSet atomicV = (AtomicValueSet)gen;
				String sqlVType = valueSetTypes.get(atomicV);
				sqlStr += atomicV.getName() + " (term " + sqlVType + ", PRIMARY KEY(term));";
			}

			else if(gen instanceof ConceptAttributeRange) {

				ConceptAttributeRange CARange = (ConceptAttributeRange)gen;
				sqlStr =
getDirectMappedSetTableCreationStatement(CARange.getConceptAttribute(), valueSetTypes);
			}

			else if(gen instanceof RoleAttributeRange) {

				RoleAttributeRange RARange = (RoleAttributeRange)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(RARange.getRoleAttribute(),
valueSetTypes);
			}

			else if(gen instanceof NegatedBasicValueSet) {

				NegatedBasicValueSet negV = (NegatedBasicValueSet)gen;
				sqlStr = getDirectMappedSetTableCreationStatement(negV.getNegatedValueSet(),
valueSetTypes);

			}

			else if(gen instanceof ConceptAttribute) {
```

```
				ConceptAttribute concAttr = (ConceptAttribute)gen;
				String sqlVType = 
valueSetTypes.get(this.getDomainFactory().getConceptAttributeRange(concAttr));
				sqlStr += concAttr.getName() +
				" (term1 " + sqlObjectType + ", term2 " + sqlVType + ", " +
				"PRIMARY KEY(term1, term2));";
			}

			else if(gen instanceof NegatedConceptAttribute) {

				NegatedConceptAttribute negConcAttr = (NegatedConceptAttribute)gen;
				sqlStr = 
getDirectMappedSetTableCreationStatement(negConcAttr.getNegatedConceptAttribute(), valueSetTypes);
			}

			else if(gen instanceof RoleAttribute) {

				RoleAttribute roleAttr = (RoleAttribute)gen;
				String sqlVType = 
valueSetTypes.get(this.getDomainFactory().getRoleAttributeRange(roleAttr));
				sqlStr += roleAttr.getName() +
				" (term1 " + sqlObjectType + ", term2 " + sqlObjectType + ", term3 " + 
sqlVType + ", " +
				"PRIMARY KEY(term1, term2, term3));";
			}

			else {				// NegatedRoleAttribute

				NegatedRoleAttribute negRoleAttr = (NegatedRoleAttribute)gen;
				sqlStr = 
getDirectMappedSetTableCreationStatement(negRoleAttr.getNegatedRoleAttribute(), valueSetTypes);
			}

			return sqlStr;
		}

		public String getEmptyTable(int arity) {
			String attributes = "";
			for (int i=0;i<arity;i++){
				attributes+="table_name";
				if (i!=arity-1) attributes+=",";
			}
			return "SELECT "+attributes+ " FROM information_schema.tables WHERE 0=1";
		}

		public String getEmptyTable(String[] aliasTerms) {
			int arity = aliasTerms.length;
			String attributes = "";
			for (int i=0;i<arity;i++){
				attributes+="table_name" + " AS " + aliasTerms[i];
				if (i!=arity-1) attributes+=",";
			}
			return "SELECT "+attributes+ " FROM information_schema.tables WHERE 0=1";

		}

		public Set<String> getListOfActiveViews() throws SQLException,
				DataLevelConfigurationException, DataLevelSQLError {
			if (this.connection!=null) {
				Set<String> setOfActiveViews = new HashSet<String>();
				PreparedStatement getViewsName = 
this.getConnection().prepareStatement(this.getViewsName);
				getViewsName.setString(1, this.getSchemaDbName());
				ResultSet rs = getViewsName.executeQuery();
				while(rs.next()) {
					String viewName = rs.getString(1).toLowerCase();
					setOfActiveViews.add(viewName);
				}
				return setOfActiveViews;
			}
			else return null;
		}

		public String getSchemaDbName() throws SQLException {
			if (this.connection!=null) return this.connection.getCatalog();
			else return null;
		}
```

```
	public HashMap<String, String> getTypeConversionTable() {
		HashMap<String, String> map = new HashMap<String, String>();
		map.put("Object", "VARCHAR(255)");
		map.put("TopD", "VARCHAR(255)");
		map.put("xs:int", "INTEGER");
		map.put("xs:short", "SMALLINT");
		map.put("xs:string", "VARCHAR(255)");
		map.put("xs:date", "DATE");
		return map;
	}

	public void setJDBCUrl(String url_no_db, String dbname) throws
DataLevelConfigurationException {
		if ((url_no_db == null) || (url_no_db.equals("")) || (dbname == null) ||
(dbname.equals("")))
				throw new DataLevelConfigurationException("JDBC Driver cannot be
empty!");
		if (url_no_db.endsWith("/")) {
			this.url = url_no_db + dbname;
		}
		else {
			this.url = url_no_db + "/" + dbname;
		}
	}
}
```

Test effettuati

In questo capitolo verrà spiegato tutto l'apparato utilizzato per l'esecuzione dei test, in modo tale da poter fare dei confronti prestazionali con eventualmente altri test effettuati con altre dotazioni. Verranno inoltre mostrate le query sulle quali sarà fatto il test ed infine saranno mostrati i risultati dei test con dei grafici che illustreranno in modo più chiaro i risultati.

Premesse

Per tutti i test si è cercato di creare le stesse condizioni per rendere il tutto più verosimile; per tutti i test i processi in esecuzione al momento del calcolo delle prestazioni erano gli stessi, il che vuol dire con buona approssimazione che le risorse a disposizione del PC utilizzato per i test erano pressoché le stesse. Inoltre durante il test gli unici processi che avevano una priorità maggiore (tale configurazione è stata settata manualmente) erano la java.exe (per la java virtual machine) e eclipse.exe (per l'ambiente di sviluppo usato per avviare QuOnto per il test).
I test sono stati effettuati utilizzando la seguente macchina:

Nome SO	Microsoft Windows XP Home Edition
Versione	5.1.2600 Service Pack 2 Build 2600
Produttore SO	Microsoft Corporation
Produttore sistema	ASUSTeK Computer Inc.
Modello sistema	A6VA
Tipo sistema	PC basato su X86
Processore	Intel Pentium M730; x86 Family 6 Model 13 Stepping 8 GenuineIntel ~1197 Mhz
Versione/data	BIOS American Megatrends Inc. A6VAAS.212, 20/02/2006
Versione SMBIOS	2.3
Directory Windows	C:\WINDOWS
Directory System	C:\WINDOWS\system32
Periferica di avvio	\Device\HarddiskVolume2
Hardware Abstraction Layer	Versione = "5.1.2600.2180 (xpsp_sp2_rtm.040803-2158)"
Memoria fisica totale	DDR2 PC2-4300 (266 MHz) 512,00 MB
Memoria fisica disponibile	129,79 MB
Memoria virtuale totale	2,00 GB
Memoria virtuale disponibile	1,94 GB
Spazio file di paging	1,22 GB
File di paging	C:\pagefile.sys
L1 data cache	32 KBytes; 8-way set associative, 64-byte line size
L1 instruction cache	32 KBytes; 8-way set associative, 64-byte line size
L2 cache	2048 Kbytes; 8-way set associative, 64-byte line size
FSB	400 MHz

Durante i test i dati erano memorizzati in una partizione secondaria dell'hard disk, quindi per eventuali confronti con altri risultati di test bisognerà tener conto anche di questo aspetto, che molto probabilmente influirà in modo negativo sulle prestazioni; però tale configurazione si è resa necessaria causa la mancanza di spazio fisico sulla partizione primaria.

Ontologie usate nei test

Prima di effettuare i test sono state effettuate alcune operazioni per ottenere delle ontologie sulle quali fare test. Siamo partiti dall'ontologia Lehigh University BenchMark (LUBM) (sviluppata a sua volta all'interno di una suite di benchmarking che potesse permettere di valutare i Semantic

Web repositories o Knowledge Base Systems in modo standard e sistematico), per poi realizzare con opportune trasformazioni l'ontologia sulla quale effettuare il test. Per quanto riguardala TBox è stata realizzata trasformando la TBox scritta in formato OWL in una TBox scritta in formato XML, secondo la DTD di QuOnto, ed eliminando tutto ciò che non può essere espresso in DL-Lite$_A$; invece per quanto riguarda la realizzazione delle Aboxes è stato utilizzato un tool scritto interamente in Java in grado di tradurre automaticamente le Aboxes scritte in formato OWL in Aboxes scritte in formato XML, secondo la DTD definita in QuOnto. Questo lavoro è stato fatto sia per SQLite sia per Derby. Le Abox sono di dimensioni differenti il che comporta il fatto di avere differenti livelli estensionali, e la dimensione è misurata in numero di università prese in considerazione; infatti è stato fatto test su una ABox relativa ad una università, una relativa a 5 università, poi 10 università ed infine 30 università. La traduzione di tali ABoxes è stata effettuata con una macchina molto più potente di quella utilizzata per i test, causa l'improponibilità dovuta all'eccessivo tempo di calcolo. Al fine di ottenere il massimo dal query answering sono stati creati degli indici per i dati estensionali utilizzati nei test, anche questa operazione come le altre descritte in precedenza sono state effettuate off-line.

Query usate nei test

Per quanto riguarda la fase di testing, sono state utilizzate14 queries fornite all'interno della suite di benchmarking della LUBM, in modo tale da permettere un raffronto eventualmente con altri reasoner. Di seguito sono riportate le 14 queries scritte nel linguaggio del calcolo relazionale:

1. {x | GraduateStudent(x) AND takesCourse(x,"Dep0.Univ0/GraduateCourse0")}
(query con grande input, alta selettività, tale query riguarda una classe e una proprietà, non assume nessuna informazione gerarchica)

2. {x,y,z | GraduateStudent(x) AND University(y) AND Department(z) AND subOrganizationOf(z,y) AND memberOf(x,z) AND undergraduateDegreeFrom(x,y)}
(query con pattern triangolare, riguarda 3 classi e tre proprietà, quindi molti join)

3. {x | Publication(x) AND publicationAuthor(x,"Dep0.Univ0/AssistantProfessor0")}
(query simile alla 1, ma Publication ha una gerarchia più ampia)

4. {x,y1,y2,y3 | Professor(x) AND worksFor(x,"Dep0.Univ0") AND name(x,y1) AND emailAddress(x,y2) AND telephone(x,y3)}
(query con 3 attributi di concetto, ampia gerarchia per Professor, piccolo input e alta selettività)

5. {x | Person(x) AND memberOf(x,"Dep0.Univ0")}
(Person e memberOf hanno una gerarchia molto ampia)

6. {x | Student(x)}
(query con grande quantità in input, e una discreta gerarchia determinata da Student, tale query è relativa ad una sola classe; assume sia la SubClassOf esplicita della relazioni tra UndergraduateStudent e Student e quella implicita tra GraduateStudent e Student)

7. {x,y | Student(x) AND Course(y) AND takesCourse(x,y) AND teacherOf("Dep0.Univ0/AssociateProfessor0",y)}
(query simile alla precedente ma con selettività più alta, infatti è relativa a 2 classi e ad una proprietà)

8. {x,y,z | Student(x) AND Department(y) AND memberOf(x,y) AND subOrganizationOf(y,"Univ0") AND emailAddress(x,z)}
(query ancora più complessa della 7 con aggiunta di un'altra proprietà)

9. {x,y,z | Student(x) AND Faculty(y) AND Course(z) AND advisor(x,y) AND teacherOf(y,z) AND takesCourse(x,z)}
(query con pattern triangolare, simile alla 2)

10. {x | Student(x) AND takesCourse(x,"Dep0.Univ0/GraduateCourse0")}
(si differenzia dalle query 6,7,8 e 9 per il fatto che richiede solo la SubClassOf implicita della relazione tra GraduateStudent e Student)

11. {x | ResearchGroup(x) AND subOrganizationOf(x,"Univ0")}
(in questa query la proprietà subOrganizationOf e stata definita transitiva, cioè le istanze di ResearchGroup sono state dichiarate come una sottoorganizzazione di un dipartimento in particolare e a sua volta sottoorganizzazione di un università in particolare, quindi per rispondere a questa query è richiesto di inferire sulla sottoorganizzazione della relazione tra le istanze di ResearchGroup e University)

12. {x,y | Chair(x) AND Department(y) AND worksFor(x,y) AND subOrganizationOf(y,"Univ0")}
(tale query richiede di inferire che professore è una istanza della classe Chair perché lui o lei sono a capo del dipartimento; query con piccolo input)

13. {x | Person(x) AND hasAlumnus("Univ0",x)}
(query di verifica per relazioni inverse. hasAlumnus è stata definita come proprietà inversa di degreeFrom la quale ha tre sottopropietà undergraduateDegreeFrom, mastersDegree-From, e doctoralDegreeFrom; quindi una persona è considerata un alunno di un università se usa una delle tre sottopropietà anziche hasAlumnus. Quindi questa query assume la relazione subPropertyOf tra degreeFrom e le sue sottoproprietà e quindi richiede di inferire su inverseOf.)

14. {x | UndergraduateStudent(x)}
(è la query più semplice: grande quantità in input, bassa selettività, assenza di gerarchia, assenza di inferenza)

Per descrivere le suddette query sono stati utilizzati dei fattori quali:

- Grandezza input: misura la porzione delle istanze delle classi coinvolte nella query sul totale delle istanze delle classi. Input grande sarà considerato una porzione superiore al 5%, basso input altrimenti.
- Selettività: misura come la porzione stimata delle istanze delle classi coinvolte nella query soddisfano i criteri di selezione della query. Si parla di alta selettività se la porzione è inferiore al 10%, bassa selettività altrimenti. Se la selettività è alta o bassa, può dipendere dal dataste utilizzato.
- Complessità: il numero di classi e di proprietà coinvolte nella query determina la complessità della stessa.
- Assunzione di gerarchia: considera se informazioni provenienti da una gerarchia di classi o da una gerarchia di proprietà sono richieste per raggiungere la risposta completa.
- Assunzione di inferenza logica: considera se l'inferenza logica è richiesta per raggiungere la completezza della risposta.

Come avviene il test

Il test viene effettuato prima per una università, poi per 5, 10 ed infine 30 università. Per ogni livello estensionale vengono valutate tutte e 14 le query e per ognuna delle quali, utilizzando come riferimento l'orologio di sistema, viene misurato il tempo di espansione della query "EXPANSION TIME" e il tempo di valutazione "EVALUATION TIME", inoltre viene valutato il numero di query congiuntive espanse all'interno della union congiuntive query "NUMERO DI CQs DENTRO LA UCQ ESPANSA", numero di risposte trovate e in quanto tempo. Il tempo totale impiegato per la risposta è calcolato come la somma dell' "EXPANSION TIME" e dell' "EVALUATION TIME".

Dati e grafici

Di seguito riportiamo i risultati dei test effettuati sia per SQLite sia per Derby con la macchina descritta nelle premesse, mostrando dati e grafici.

SQLite

Riportiamo ora tutti i dati relativi ad i test effettuati con QuOnto utilizzando il driver per SQLite. Iniziamo col riportare delle tabelle relative ad ogni singola query per mostrare tutti i parametri misurati che la riguardano.

Query 1

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	121	1	12	121
5 università	0	199	1	6	199
10 università	0	261	1	6	261
30 università	0	390	1	22	390

Query 2

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	281	11125	26	103	11406
5 università	250	54750	26	492	55000
10 università	250	106484	26	879	106734
30 università	266	421250	26	2761	421516

Query 3

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	16	1	7	16
5 università	0	47	1	9	47
10 università	0	125	1	9	125
30 università	0	156	1	10	156

Query 4

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	78	94	36	21	172
5 università	78	391	36	18	469
10 università	78	422	36	13	500

30 università	78	672	36	14	750

Query 5

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	15	16	10	739	31
5 università	31	31	10	796	62
10 università	15	32	10	732	47
30 università	15	141	10	739	156

Query 6

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	2078	11	13570	2078
5 università	0	12547	11	57017	12547
10 università	0	25812	11	110768	25812
30 università	0	345812	11	354125	345812

Query 7

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	46	2	45	46
5 università	0	78	2	40	78
10 università	0	109	2	43	109
30 università	47	5813	2	95	5860

Query 8

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	2063	250562	186	13320	252625
5 università	2031	144110	186	13320	146141
10 università	2063	153719	186	13320	155782
30 università	2281	181562	186	13320	183843

Query 9

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	1782	2	184	1782
5 università	0	12016	2	851	12016
10 università	15	34469	2	1640	34484
30 università	0	296453	2	15623	296453

Query 10

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	0	1	12	0
5 università	0	0	1	6	0

10 università	0	31	1	6	31
30 università	10	881	1	22	891

Query 11

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	16	78	3	200	94
5 università	16	62	3	200	78
10 università	0	171	3	200	171
30 università	44	1753	3	200	1797

Query 12

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	15	5578	10	20	5593
5 università	0	3375	10	20	3375
10 università	15	3610	10	20	3625
30 università	78	7364	10	20	7422

Query13

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	16	93	5	161	109
5 università	16	515	5	757	531
10 università	16	969	5	1643	985
30 università	94	28531	5	4792	28625

Query14

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	0	1	10280	0
5 università	0	0	1	43367	0
10 università	0	0	1	82158	0
30 università	0	312	1	269191	312

Ora riportiamo dei grafici che mettono meglio in mostra l'aumento di complessità del query answering in relazione all'aumento della grandezza dei dati e al tipo di query su cui viene fatto il query answering. Per garantire una migliore visibilità dei risultati, nei grafici che riportiamo ora, saranno raggruppate le query che hanno mostrato un tempo totale di calcolo almeno paragonabile, in quanto mettere nello stesso grafico la query 14 (la più semplice di tutte perché grande quantità in input, bassa selettività e assenza di gerarchia) che ha un tempo totale di valutazione per 30 università dell'odine delle centinaia di millisecondi e la query 8 (elevata selettività) che ha un tempo totale di valutazione per 30 università dell'ordine delle centinaia di migliaia. Quindi abbiamo preferito, per le motivazioni appena dette, mettere insieme nello stesso grafico i risultati delle query 1,3,4,5,7,10,11,14 che rappresentano queries con tempi di calcolo (sempre per 30 università) dell'ordine massimo delle migliaia, e in un altro grafico le rimanenti query 2,6,8,9,12,13 che hanno un tempo di calcolo (sempre per 30 università) che va dalle migliaia alle centinaia di migliaia. Tali

grafici avranno sull'asse delle ascisse la dimensione dei dati espressi in numero di università del livello estensionale della conoscenza, mentre sull'asse delle ordinate il tempo totale di calcolo della query espresso in millisecondi; ogni query verrà graficata di un colore differente in modo tale da favorire una migliore visione dei risultati.
Nella parte inferiore del grafico sono riportati i valori all'interno di una tabella che sono stati graficati.

Grafico tempi query leggere

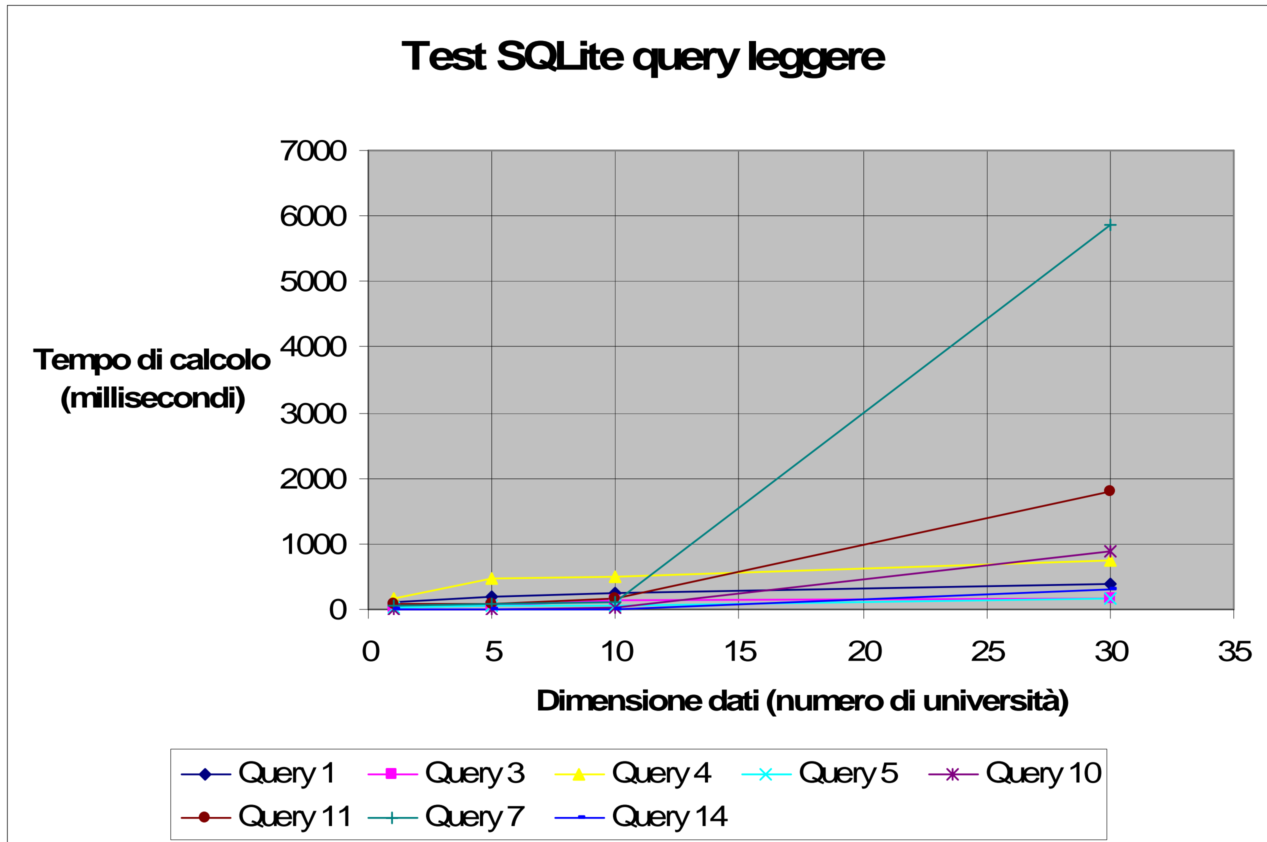


Tabella tempi query leggere

Numero Università	Query 1 (ms)	Query 3 (ms)	Query 4 (ms)	Query 5 (ms)	Query 10 (ms)	Query 11 (ms)	Query 7 (ms)	Query 14 (ms)
1	121	16	172	31	0	94	46	0
5	199	47	469	62	0	78	78	0
10	261	125	500	47	31	171	109	0
30	390	156	750	156	891	1797	5860	312

Grafico tempi query pesanti

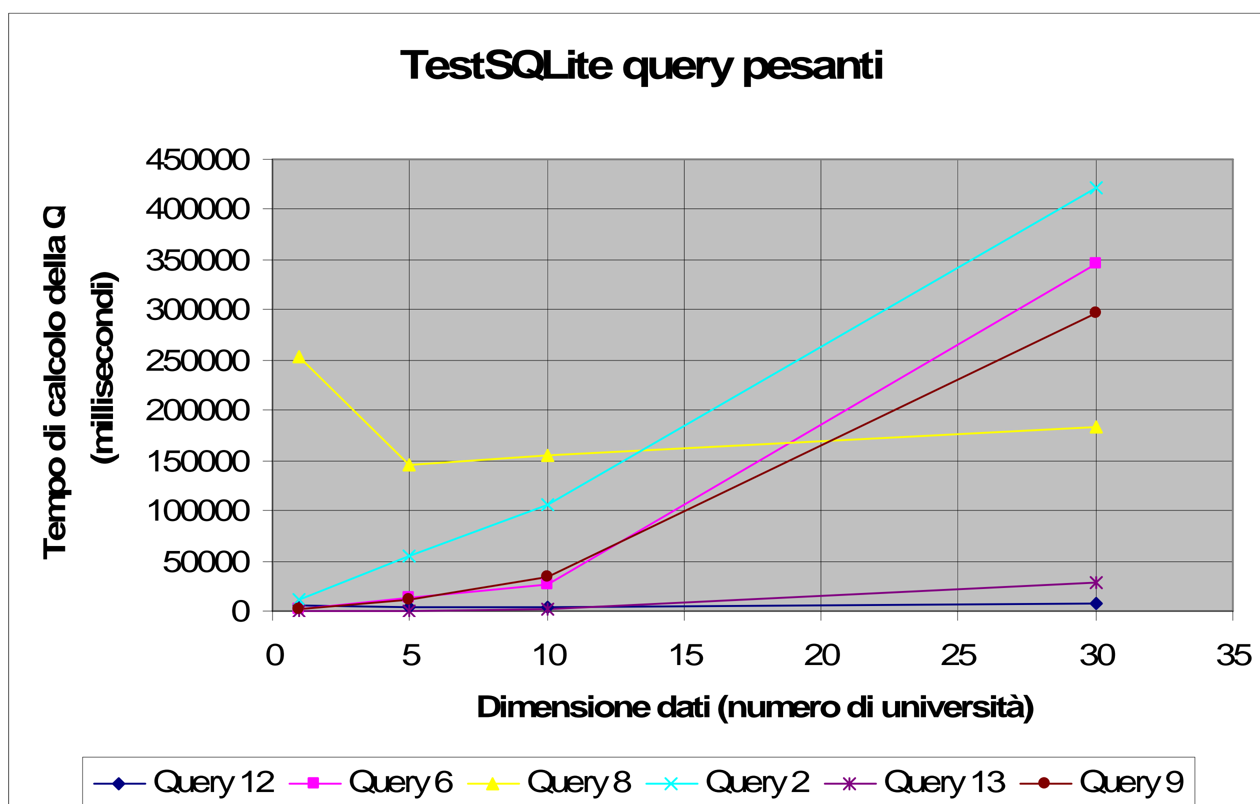


Tabella tempi query pesanti

Numero Università	Query 12 (ms)	Query 6 (ms)	Query 8 (ms)	Query 2 (ms)	Query 13 (ms)	Query 9 (ms)
1	5593	2078	252625	11406	109	1782
5	3375	12547	146141	55000	531	12016
10	3625	25812	155782	106734	985	34484
30	7422	345812	183843	421516	28625	296453

Come si può vedere dai grafici e dai dati riportati nelle tabelle, all'aumentare della dimensione dei dati i tempi di risposta si dilatano in modo approssimativamente lineare e per fortuna assolutamente non esponenziale. Questo fatto è davvero notevole, perché, se l'andamento fosse stato esponenziale il query answering sarebbe stato deleterio dal punto di vista del tempo impiegato.
In QuOnto, i fattori più importanti che determinano la complessità di una query sono fondamentalmente 2: il numero di join e il numero di union presenti nella query espansa. Se prendiamo in considerazione le queries 6 e 14, dalla struttura apparentemente simili, notiamo che hanno dei tempi di risposta davvero molto diversi fra loro. Tale differenza dipende dal fato che la query 6, parlando di studenti, si riferisce ad una gerarchia molto ampia che determina la presenza di un discreto numero di union nella query espansa che invece sono del tutto assenti nell'altra query espansa, che di fatto corrisponde ad un semplice instances retrieval senza gerarchia.

Note sul test con SQLite

Oltre a queste 14 query è stata testata un altra query di cui non sono stati riportati i risultati, tale query è la numero 15, tale query presenta un pattern pentagonale e di seguito ne riportiamo la dicitura espressa in linguaggio del calcolo relazionale:

{x | hasSameHomeTownWith(x,y) AND isMemberOf(y,z) AND hasMember(z,t) AND isCrazyAbout(t,w) AND isCrazyAbout(x,w)}

Quando si provava a fare test su questa query QuOnto andava in errore e riportiamo di seguito l'errore:
"Maximum Number Of Terms In A Compound SELECT Statement"
In realtà questo non è un errore di QuOnto, ma un errore dovuto ad un limite di SQLite, secondo cui il massimo numero di termini che possono essere messi in UNION, UNION ALL, EXCEPT, oppure INTERSECT è determinato da un parametro fisso e tale parametro di default è settato a 500, il che vuol dire che, a meno di ricompilazione del codice C di SQLite, non si può eseguire una query con più di 500 congiunctive queries (CQs). La query 15, che è la più pesante, è composta infatti di 1143 congiunti, ecco spiegato il motivo di tale errore. Per maggiori informazioni sui limiti di SQLite è possibile visitare il seguente sito internet: http://www.sqlite.org/limits.html .

Derby

Riportiamo ora tutti i dati relativi ad i test effettuati con QuOnto utilizzando il driver per Derby. Iniziamo, come precedentemente fatto con SQLite, col riportare le tabelle relative ad ogni singola query per mostrare tutti i parametri misurati che la riguardano.

Query 1

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	93	1	12	93
5 università	0	375	1	6	375
10 università	0	523	1	6	523
30 università	0	579	1	22	579

Query 2

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	297	8890	26	103	9187
5 università	282	11609	26	492	11891
10 università	328	23031	26	879	23359
30 università	305	60351	26	2761	60656

Query 3

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	16	16	1	7	32
5 università	0	93	1	9	93
10 università	0	125	1	9	125
30 università	16	187	1	10	203

Query 4

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	78	4454	36	21	4532
5 università	78	5047	36	18	5125
10 università	78	5235	36	13	5313
30 università	62	5500	36	14	5562

Query 5

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	15	79	10	739	94
5 università	16	94	10	736	110
10 università	16	109	10	732	125
30 università	16	125	10	739	141

Query 6

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	78515	11	13570	78515
5 università	0	1202641	11	57017	1202641
10 università	16	5092875	11	110768	5092891
30 università	15	111668954	11	354125	111668969

Query 7

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	31	2	45	31
5 università	0	438	2	40	438
10 università	0	2594	2	43	2594
30 università	18	5011	2	95	5029

Query 8

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	2016	189563	186	13320	191579
5 università	2156	179688	186	13320	179688
10 università	2016	188218	186	13320	190234
30 università	2485	198500	186	13320	200985

Query 9

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	844	2	184	844
5 università	16	12781	2	851	12797
10 università	0	27187	2	1640	27187
30 università	0	158797	2	15623	158797

Query 10

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	15	1	12	15
5 università	0	19	1	6	19
10 università	0	20	1	6	20
30 università	0	94	1	22	94

Query 11

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME

1 università	0	94	3	200	94
5 università	15	391	3	200	406
10 università	16	781	3	200	797
30 università	15	901	3	200	916

Query 12

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	16	862	10	20	878
5 università	16	609	10	20	625
10 università	16	968	10	20	984
30 università	16	1328	10	20	1344

Query 13

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	297	5	161	297
5 università	0	609	5	757	609
10 università	16	1375	5	1643	1391
30 università	15	1875	5	4792	1890

Query 14

	EXPANSION TIME	EVALUATION TIME	NUMERO DI CQs DENTRO LA UCQ ESPANSA	ANSWERS FOUND	TOTAL TIME
1 università	0	14875	1	10280	14875
5 università	0	262860	1	43367	262860
10 università	0	1181765	1	82158	1181765
30 università	0	28509265	1	269191	28509265

Ora, come già fatto per SQLite, riportiamo dei grafici che mettono in risalto l'aumento di complessità del query answering in relazione all'aumento della grandezza dei dati e al tipo di query su cui viene fatto il query answering. Per garantire una migliore visibilità dei risultati anche per Derby, nei grafici che riportiamo ora, saranno raggruppate le query che hanno mostrato un tempo totale di calcolo almeno paragonabile. Quindi sono stati messi insieme nello stesso grafico i risultati delle query 1,3,4,5,7,10,11,12,13 che rappresentano queries con tempi di calcolo (per 30 università) dell'ordine massimo delle migliaia; in un altro grafico sono state riportate le query 2,8,9 che hanno un tempo di calcolo (sempre per 30 università) che va dalle decine di migliaia alle centinaia di migliaia; infine in un altro grafico sono stati riportati i tempi relativi alle query 6,14 che hanno avuto bisogno di un tempo di calcolo (sempre per 30 università) che va dalle decine di milioni di millisecondi alle centinaia di milioni di millisecondi. Tali grafici , come già spiegato in precedenza per SQLite, avranno sull'asse delle ascisse la dimensione dei dati espressi in numero di università del livello estensionale della conoscenza, mentre sull'asse delle ordinate il tempo totale di calcolo della query espresso in millisecondi; ogni query verrà graficata di un colore differente in modo tale da favorire una migliore visione dei risultati.
Nella parte inferiore del grafico è riportata una tabella con i valori che sono stati graficati.

Grafico tempi query leggere

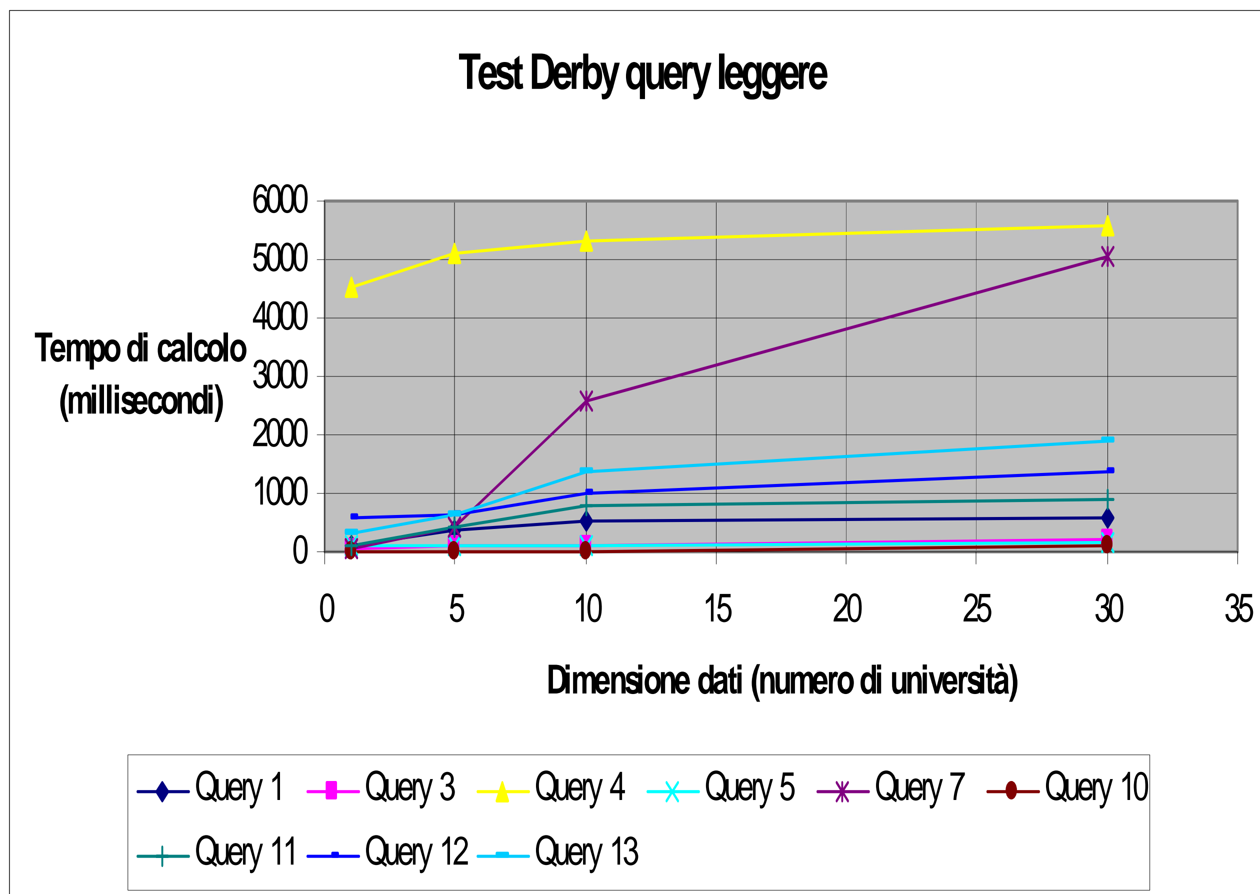


Tabella tempi query leggere

Numero Università	Query 1 (ms)	Query 3 (ms)	Query 4 (ms)	Query 5 (ms)	Query 7 (ms)	Query 10 (ms)	Query 11 (ms)	Query 12 (ms)	Query 13 (ms)
1	93	32	4532	94	31	15	94	578	297
5	375	93	5125	110	438	19	406	625	609
10	523	125	5313	125	2594	20	797	984	1391
30	579	203	5562	141	5029	94	916	1344	1890

Grafico tempi query pesanti

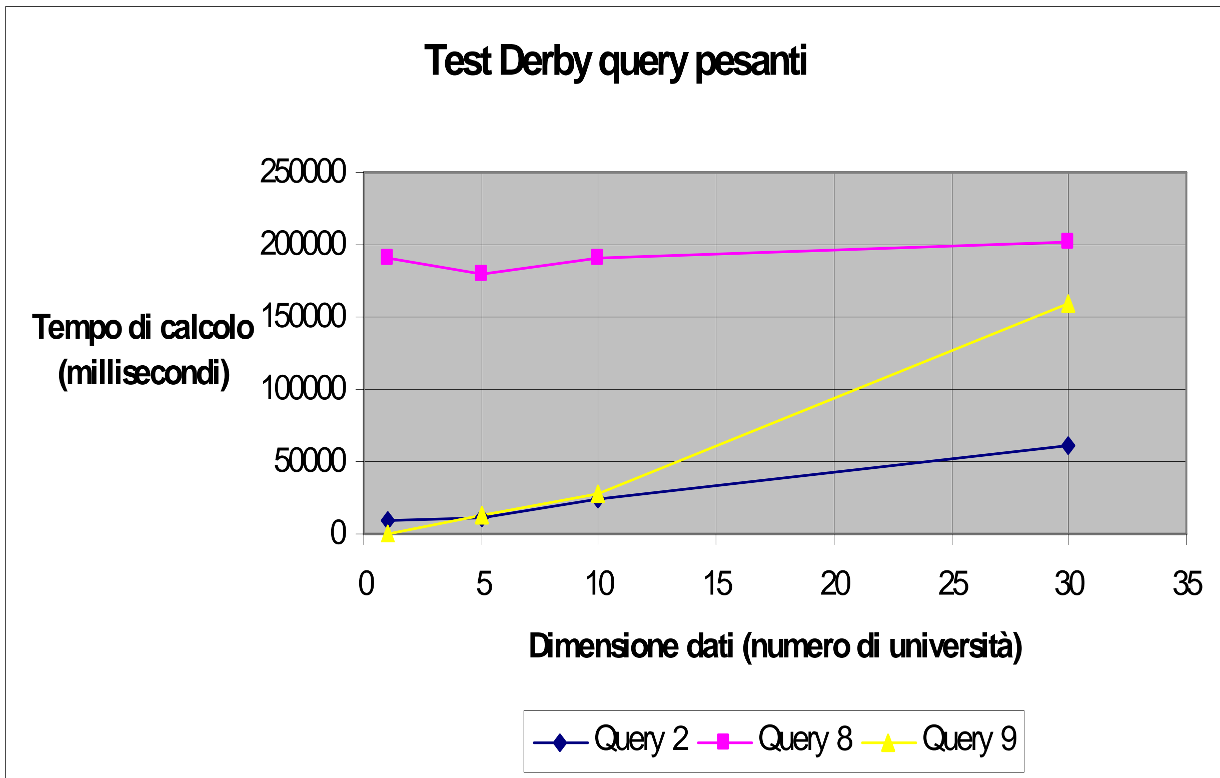


Tabella tempi query pesanti

Numero università	Query 2 (ms)	Query 8 (ms)	Query 9 (ms)
1	9187	191579	844
5	11891	179688	12797
10	23359	190234	27187
30	60656	200985	158797

Grafico tempi query molto pesanti

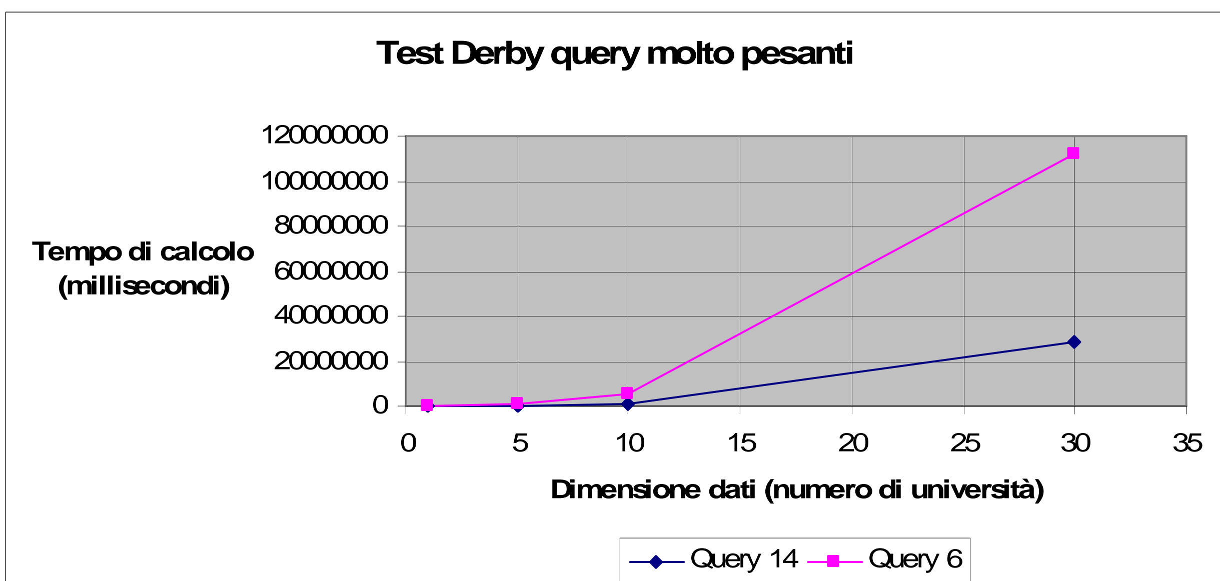


Tabella tempi query molto pesanti

Numero università	Query 14 (ms)	Query 6 (ms)
1	14875	78515
5	262860	1202641
10	1181765	5092891
30	28509265	111668969

Come si può notare dai grafici e dai dati riportati nelle tabelle, all'aumentare della dimensione dei dati i tempi di risposta non sempre si dilatano si dilatano in modo approssimativamente lineare, come nel caso della query 6, dove si passa da 78515 millisecondi (poco più di un minuto) per una sola università e si arriva a ben 111668969 millisecondi (circa 31 ore) per 30 università. La stessa osservazione fatta in precedenza per le query 6 e 14 in SQLite vale anche in questo caso per Derby; in quanto, come già detto, in QuOnto i fattori più importanti che determinano la complessità di una query sono il numero di join e il numero di union presenti nella query espansa.

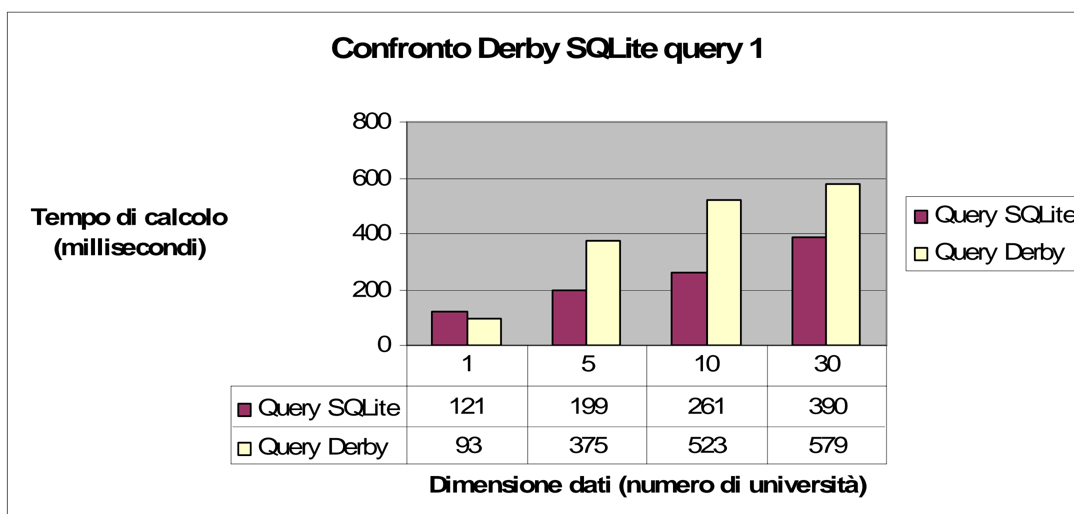
Note sul test con Derby

Anche nei test di Derby è stata testata un altra query, la numero 15, ed anche in questo caso non sono stati riportati i dati, ma non per gli stessi motivi di SQLite (limitazione sul massimo numero di termini che possono essere messi in UNION, UNION ALL, EXCEPT, oppure INTERSECT), ma in questo caso il problema è stato un altro:
"Java heap space: java.lang.OutOfMemoryError".
Tale errore si verifica quando la JVM va a lavorare fuori dalla memoria durante l'esecuzione di benchmark pesanti, come nel caso dell'esecuzione della query 17:
{x | hasSameHomeTownWith(x,y) AND isMemberOf(y,z) AND hasMember(z,t) AND isCrazyAbout(t,w) AND isCrazyAbout(x,w)}.
Anche se questo problema è risolvibile, però tale query non è stata testata in quanto molto probabilmente avrebbe impiegato tempi dell'ordine di diversi giorni, se non settimane, in quanto questa query dal punto di vista della complessità è molto più complessa della query 6 la quale ha impiegato oltre un giorno di calcolo.

Confronto prestazioni query tra Derby e SQLite

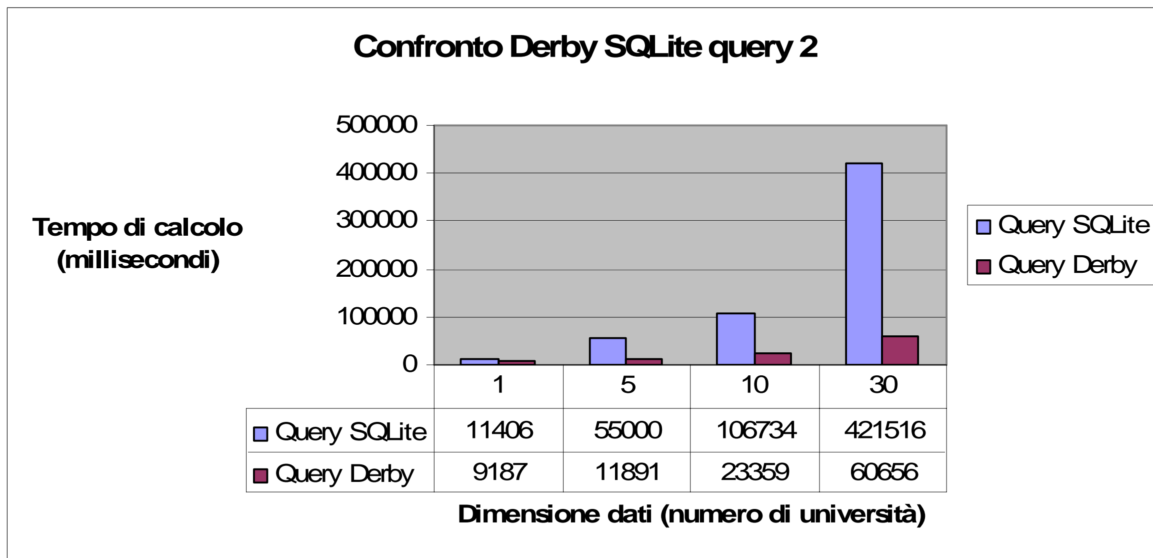
Ora verranno riportati dei grafici e delle tabelle che permetteranno di mettere a confronto le prestazioni dei due DBMS utilizzati nei test query per query.

Confronto query 1



Ricordiamo, per maggior precisione, che la query 1 espressa in linguaggio del calcolo relazionale è: {x | GraduateStudent(x) AND takesCourse(x,"Dep0.Univ0/GraduateCourse0")}.
Ora andiamo ad analizzare i comportamenti dei due DBMS in relazione a questa query. Come si può vedere dal grafico e dai dati riportati in tabella, SQLite per questa query si comporta meglio rispetto a Derby, anche se entrambi i DBMS all'aumentare della dimensione della Abox mantengono il tempo di risposta addirittura quasi costante, ossia i tempi crescono in modo meno che lineare rispetto alla crescita della Abox. Anche se Derby si dimostra un po' più lento, comunque sia si mantiene su tempi di risposta accettabili.

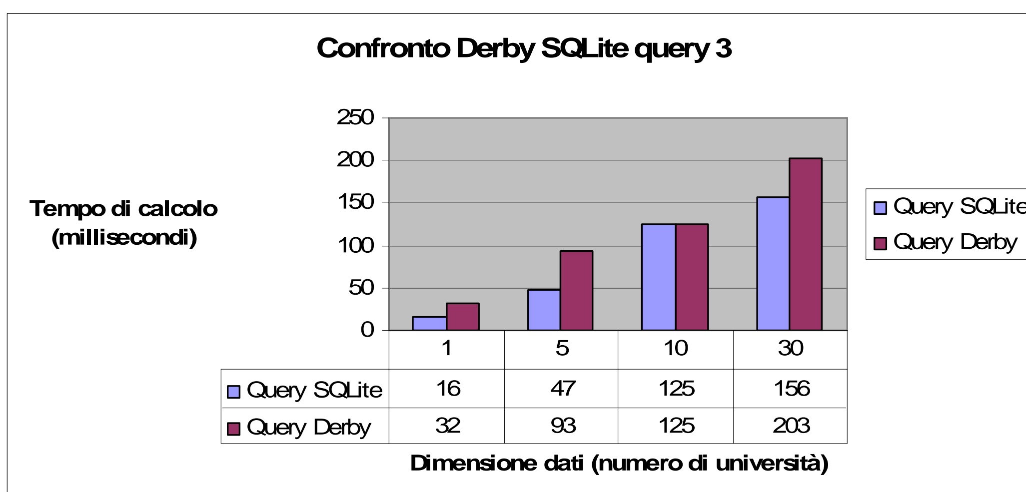
Confronto query 2



	1	5	10	30
Query SQLite	11406	55000	106734	421516
Query Derby	9187	11891	23359	60656

La query 2 espressa in linguaggio del calcolo relazionale è:
{x,y,z | GraduateStudent(x) AND University(y) AND Department(z) AND subOrganizationOf(z,y) AND memberOf(x,z) AND undergraduateDegreeFrom(x,y)}
Questa query avendo un pattern triangolare, richiedendo molte condizioni ed avendo 26 conjunctive querys nella union conjunctive query espansa risulta essere abbastanza più pesante rispetto alla precedente ed i risultati dei test lo dimostrano. Rispetto alla prima query, su questa a comportarsi meglio dal punto di vista dei tempi di risposta è Derby, che come nel caso precedente all'aumentare della dimensione della Abox i tempi di risposta aumentano in modo addirittura meno che lineare, mentre SQLite dimostra tempi di risposta abbastanza più lunghi e con un aumento circa lineare rispetto alla crescita della dimensione della Abox.

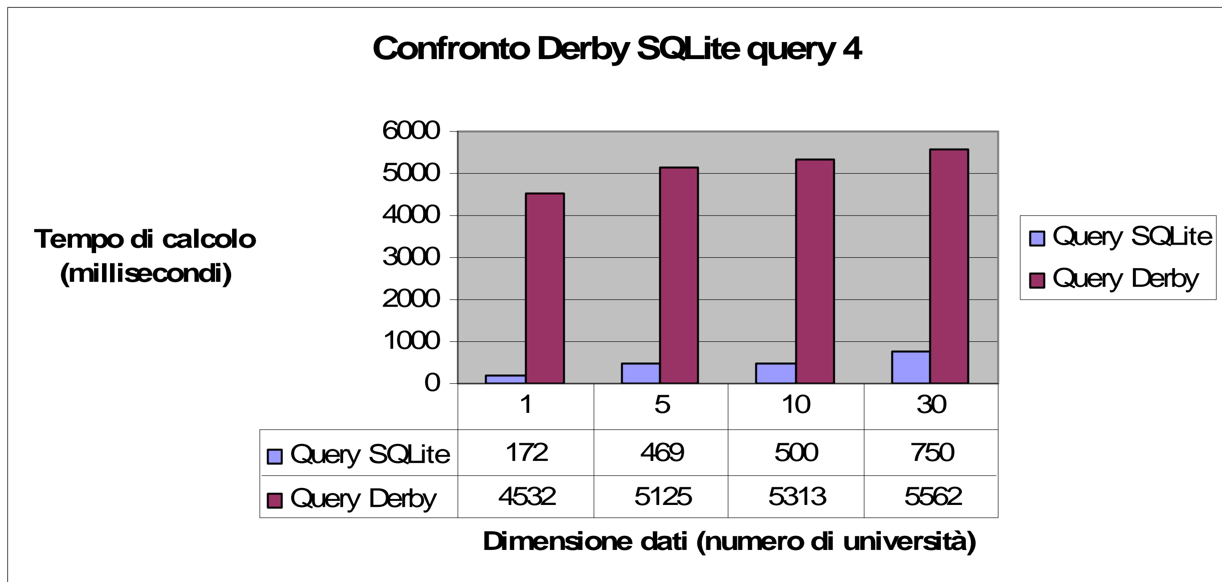
Confronto query 3



	1	5	10	30
Query SQLite	16	47	125	156
Query Derby	32	93	125	203

La query 3 espressa in linguaggio del calcolo relazionale è:
{x | Publication(x) AND publicationAuthor(x,"Dep0.Univ0/AssistantProfessor0")}
Questa query presenta una struttura simile alla query numero 1, infatti i tempi di risposta sia di Derby sia di SQLite sono confrontabili, le differenze di tempi di risposta con la query 1 sono dovute probabilmente al fatto che si va ad interrogare su una gerarchia differente. In generale per questa query il comportamento di SQLite è lievemente migliore, anche se per entrambi al crescere della Abox i tempi di risposta crescono in modo meno che lineare lineare.

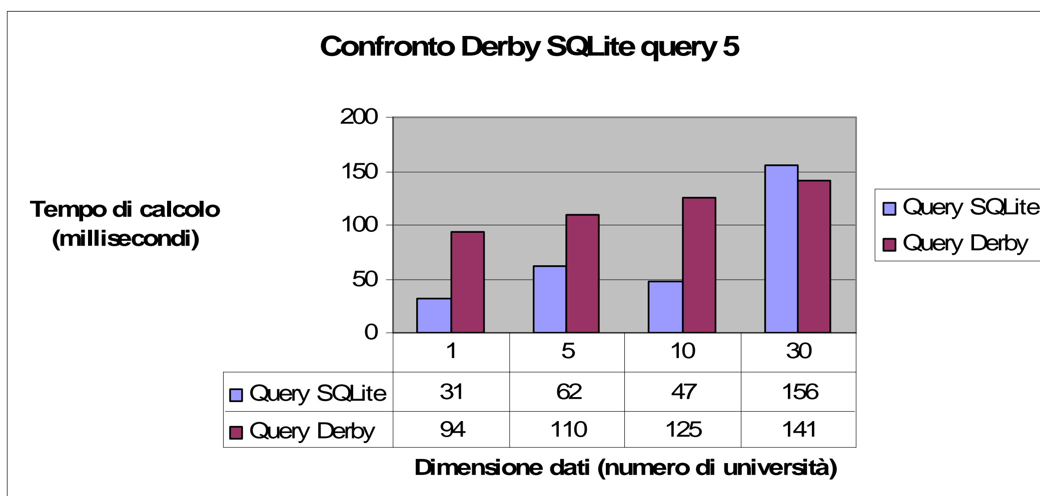
Confronto query 4



	1	5	10	30
Query SQLite	172	469	500	750
Query Derby	4532	5125	5313	5562

La query 4 espressa in linguaggio del calcolo relazionale è:
{x,y1,y2,y3 | Professor(x) AND worksFor(x,"Dep0.Univ0") AND name(x,y1) AND emailAddress(x,y2) AND telephone(x,y3)}
Questa è una query con 3 attributi di concetto, che va a lavorare su una gerarchia abbastanza ampia. QuOnto quando espande la query genera 36 conjunctive querys. Anche in questa query all'aumentare della Abox i tempi si dilatano in modo meno che lineare sia per SQLite che per Derby, solo che SQLite in questa query si dimostra molto più veloce, circa 10 volte più veloce. Quindi l'unica spiegazione di questa incredibile differenza è che SQLite sulle query in cui si richiedono attributi di concetto si comporta meglio.

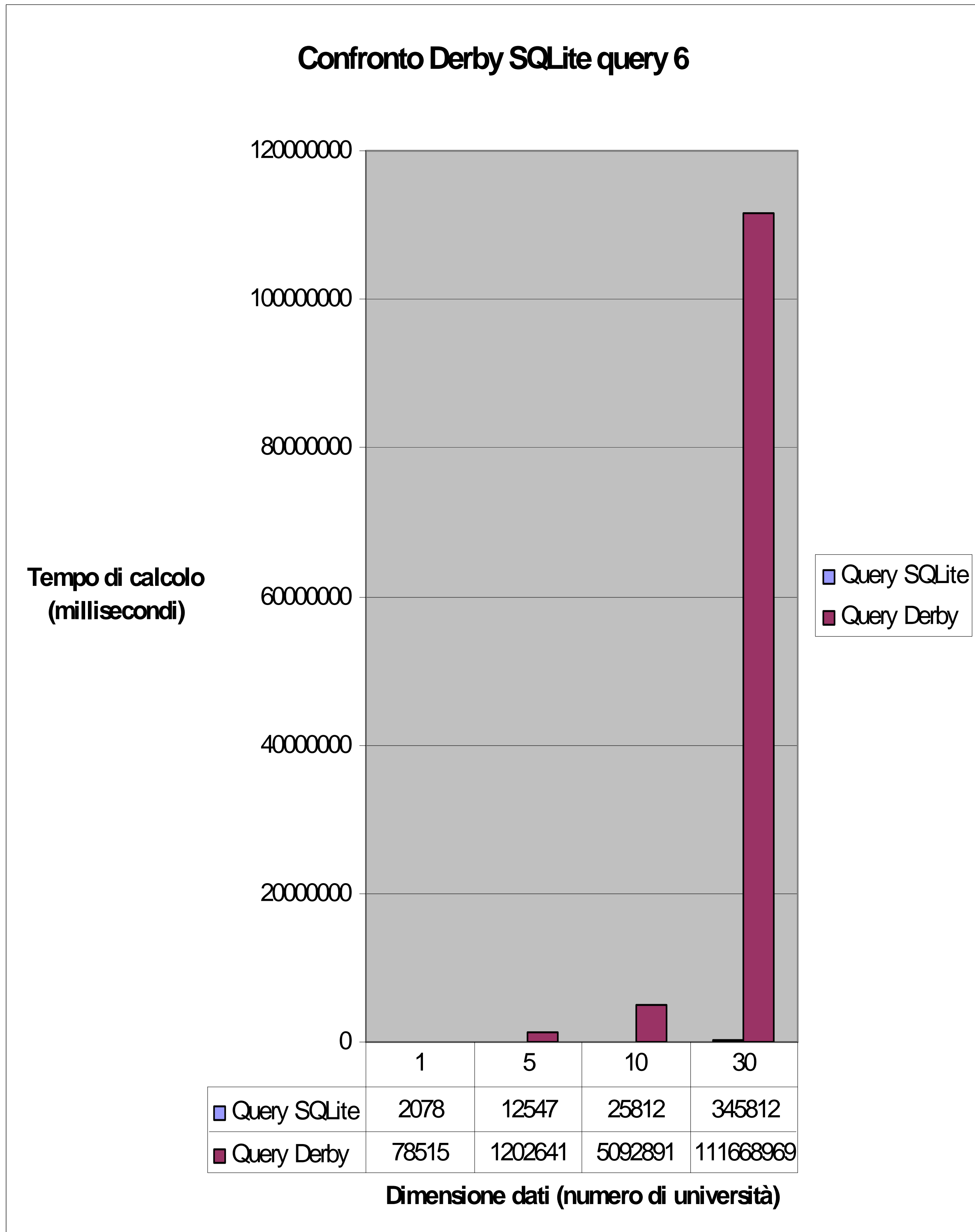
Confronto query 5



	1	5	10	30
Query SQLite	31	62	47	156
Query Derby	94	110	125	141

La query 5 espressa in linguaggio del calcolo relazionale è:
{x | Person(x) AND memberOf(x,"Dep0.Univ0")}
Questa query è molto simile alla query numero 3, ma con una gerarchia relativa a Person ancora più grande. Valgono le stesse considerazioni fatte per la query 3, anche se in questo caso per una Abox di 30 università si è comportato leggermente meglio Derby, mentre per 1,5,10 università SQLite è stato un po' più veloce.

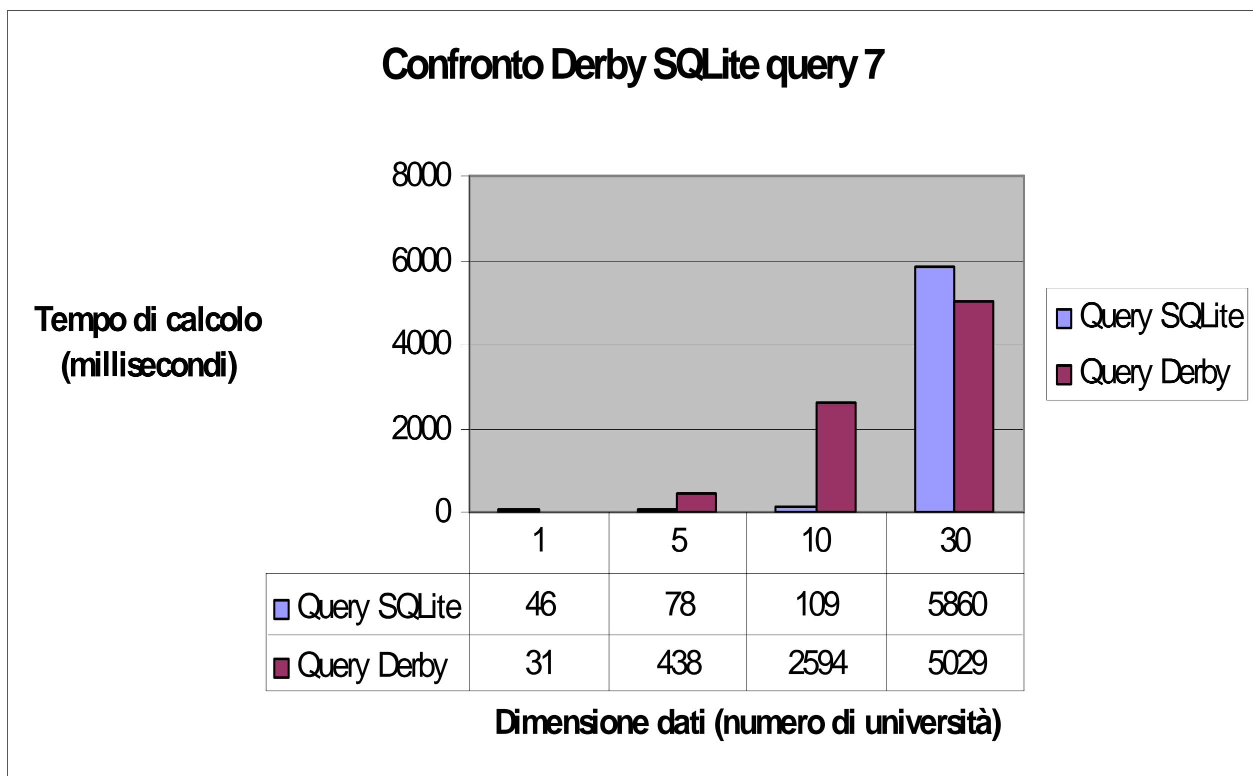
Confronto query 6



	1	5	10	30
Query SQLite	2078	12547	25812	345812
Query Derby	78515	1202641	5092891	111668969

La query 6 espressa in linguaggio del calcolo relazionale è:
{x | Student(x)}
Questa è una query molto semplice, ma con grandissima quantità in input. Come chiaramente si evince dal grafico e dai dati, le prestazioni di SQLite non sono assolutamente confrontabili con quelle di Derby, in quanto quest'ultimo per eseguire la query su una Abox di 30 università ed individuare i 354125 studenti presenti nel database ha impiegato poco più di 31 ore, mentre per eseguire la stessa query SQLite ha impiegato solamente poco meno di 6 minuti, ossia Derby e stato oltre 320 volte più lento. Dai risultati di tale query si evince che per query di questo tipo con grandi quantità di dati in ingresso e dove ci sono molti elementi da restituire visto la non selettività, Derby non è assolutamente in grado di dare risposta in tempi accettabili.

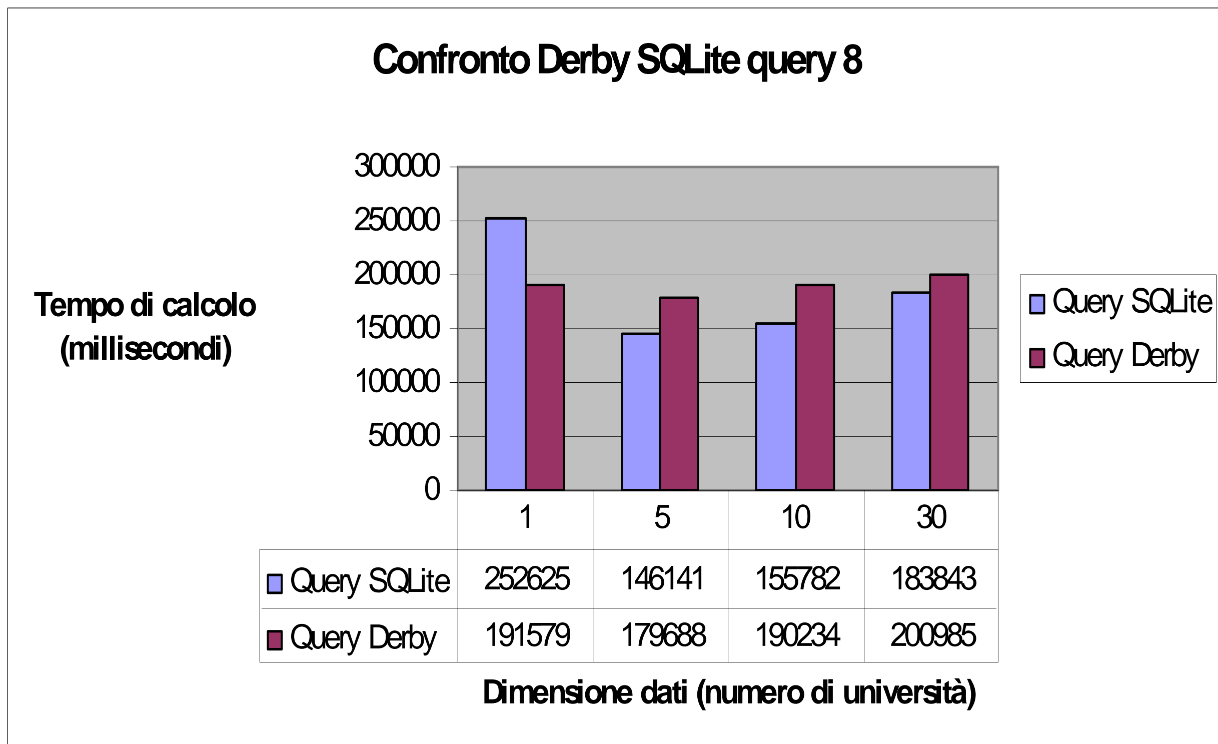
Confronto query 7



	1	5	10	30
Query SQLite	46	78	109	5860
Query Derby	31	438	2594	5029

La query 7 espressa in linguaggio del calcolo relazionale è:
{x,y | Student(x) AND Course(y) AND takesCourse(x,y) AND teacherOf("Dep0.Univ0/AssociateProfessor0",y)}
Questa query, rispetto alla precedente ha una selettività molto più elevata. Nella union conjunctive querys vi sono solo 2 conjunctive querys e grazie alla sua maggiore selettività non vi sono molti elementi da restituire (fattore che come precedentemente abbiamo visto va ad influire di molto sui tempi di risposta di Derby). Come si evince dai dati e dal grafico, per 1,5,10 università SQLite si comporta molto meglio risultando molto più veloce, mentre per il test relativo a 30 università Derby è stato un poco più veloce, comunque sia i tempi di risposta sono confrontabili, ossia non si sono viste grandi differenze di tempi di risposta.

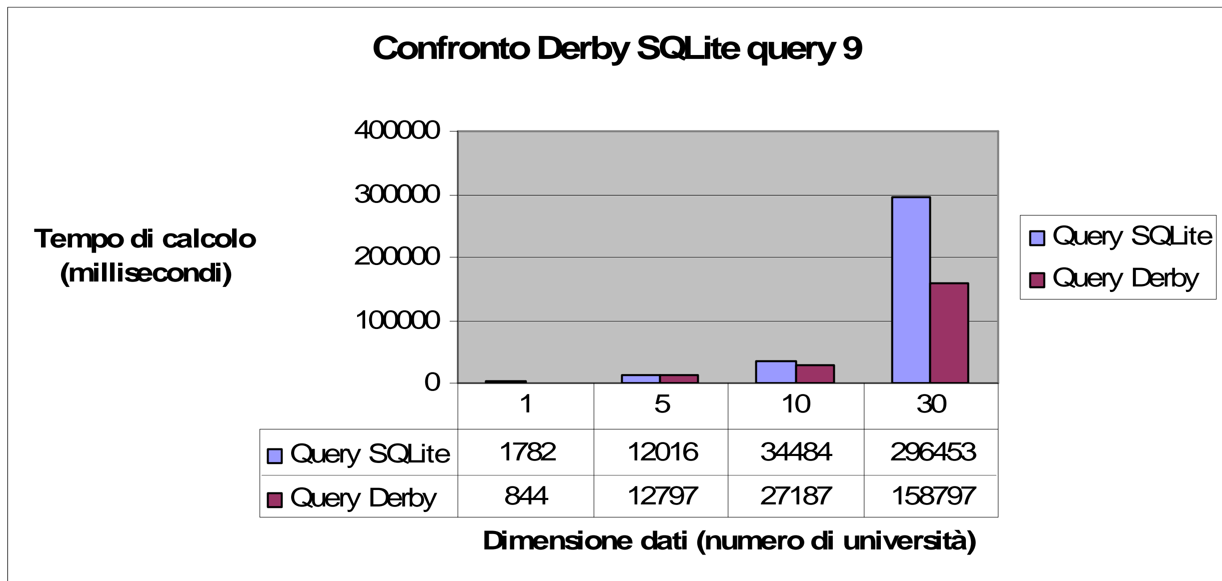
Confronto query 8



	1	5	10	30
Query SQLite	252625	146141	155782	183843
Query Derby	191579	179688	190234	200985

La query 8 espressa in linguaggio del calcolo relazionale è:
{x,y,z | Student(x) AND Department(y) AND memberOf(x,y) AND subOrganizationOf(y,"Univ0") AND emailAddress(x,z)}.
La query 8 è una query abbastanza complessa, infatti nella union conjunctive querys sono presenti ben 186 query espanse; tale query restituisce, indipendentemente dalla Abox considerata, 13200 risultati in uscita. Essendo molte le query da espandere, in questo caso il tempo impiegato per l'espansione non è nullo o quasi come nei casi precedenti, qui ci aggiriamo intorno ai 2000 millisecondi, comunque quasi trascurabile rispetto ai tempi di valutazione. Per questa query differentemente dalla maggior parte dei casi SQLite si comporta peggio per quanto riguarda la Abox da 1 università, mentre si comporta meglio per 5,10,30 università. Comunque sia i tempi di answering sono confrontabili tra Derby e SQLite. In questa query si vede un anomalia riscontrata anche in qualche altra query, ossia che per una sola università le risposte dei DBMS sono più lente rispetto ad Aboxes più grandi, probabilmente ciò sarà dovuto al fatto che, sia Derby sia SQLite, adotteranno delle ottimizzazioni solo per Abox da una determinata dimensione in poi, in quanto comunque sia, i tempi di risposta senza le supposte ottimizzazioni sono comunque accettabili. Concludo facendo notare, in questo caso solo per 5 10 30 università, il solito andamento crescente dei tempi di answering rispetto alla crescente dimensione dell'Abox.

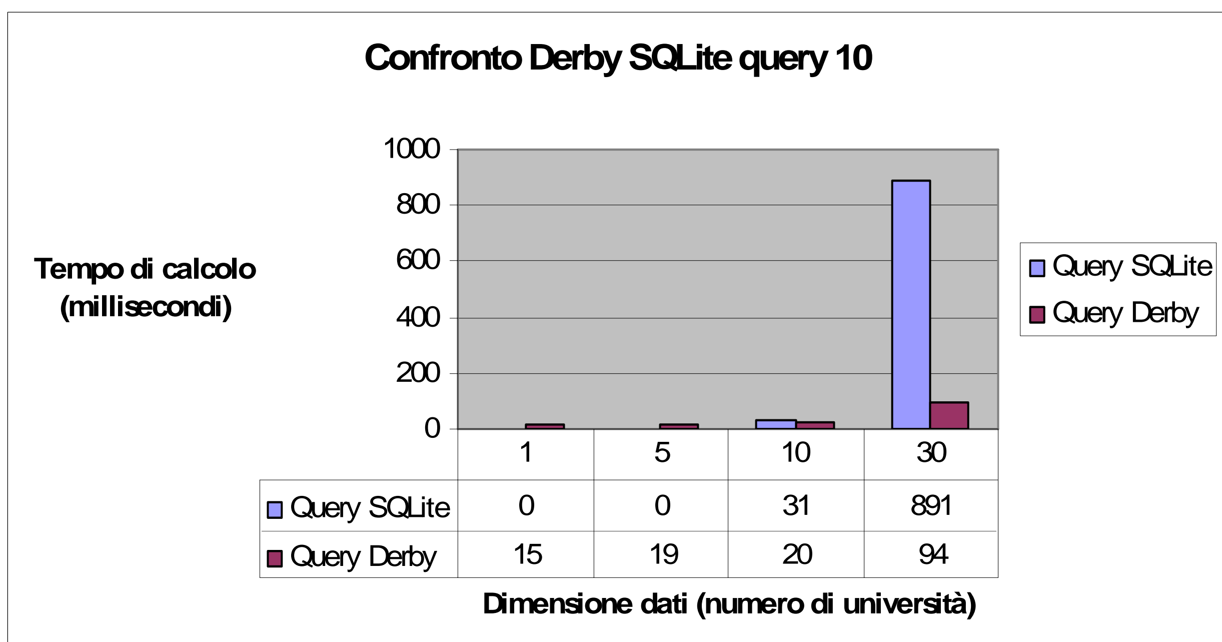
Confronto query 9



	1	5	10	30
Query SQLite	1782	12016	34484	296453
Query Derby	844	12797	27187	158797

La query 9 espressa in linguaggio del calcolo relazionale è:
{x,y,z | Student(x) AND Faculty(y) AND Course(z) AND advisor(x,y) AND teacherOf(y,z) AND takesCourse(x,z)}
Questa è una query con pattern triangolare che nella union conjunctive query espansa ha solo 2 conjunctive querys. Su questa query comportarsi meglio è Stato Derby, e la differenza la si è vista soprattutto per l'esecuzione sulle Aboxes da 1 e 30 università dove Derby è stato quasi il doppio più veloce, mentre per 5,10 università i tempi sono risultati simili. La differenza fondamentale che va presa in considerazione è quella relativa a 30 università, perché su piccole quantità di dati è possibile che un DBMS adotti delle ottimizzazioni e l'altro no, perché molto probabilmente avranno una soglia diversa dalla quale in poi adottare ottimizzazioni oppure tali differenze possono essere dovute a ritardi di diverso genere dovuti alla macchina che sta eseguendo il test visto che si tratta di pochi millisecondi.

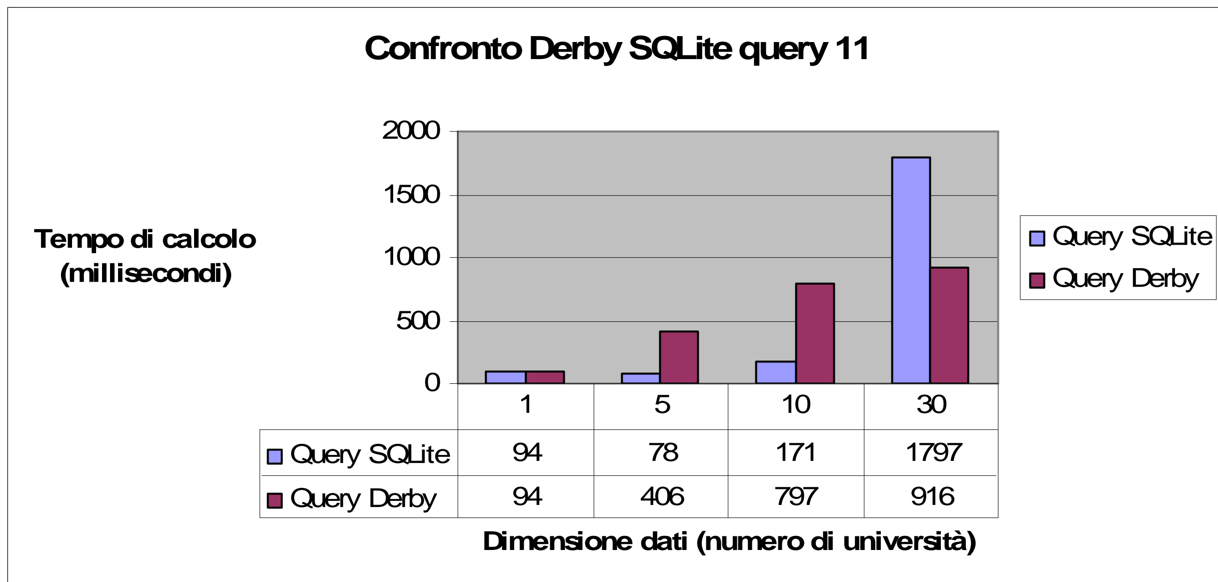
Confronto query 10



	1	5	10	30
Query SQLite	0	0	31	891
Query Derby	15	19	20	94

La query 10 espressa in linguaggio del calcolo relazionale è:
{x | Student(x) AND takesCourse(x,"Dep0.Univ0/GraduateCourse0")}
Questa è una query molto semplice, con una sola query nella query espansa, ha una selettività molto alta infatti i risultati saranno tutti gli studenti che frequentano il corso in esame. Per questa query, come già in precedenza siamo stati abituati a vedere, il comportamento di SQLite si è dimostrato migliore per quanto riguarda Aboxes di piccola taglia, mentre le sue prestazioni degradano all'aumentare della taglia della Abox (anche se non è stato sempre così per tutte le query).

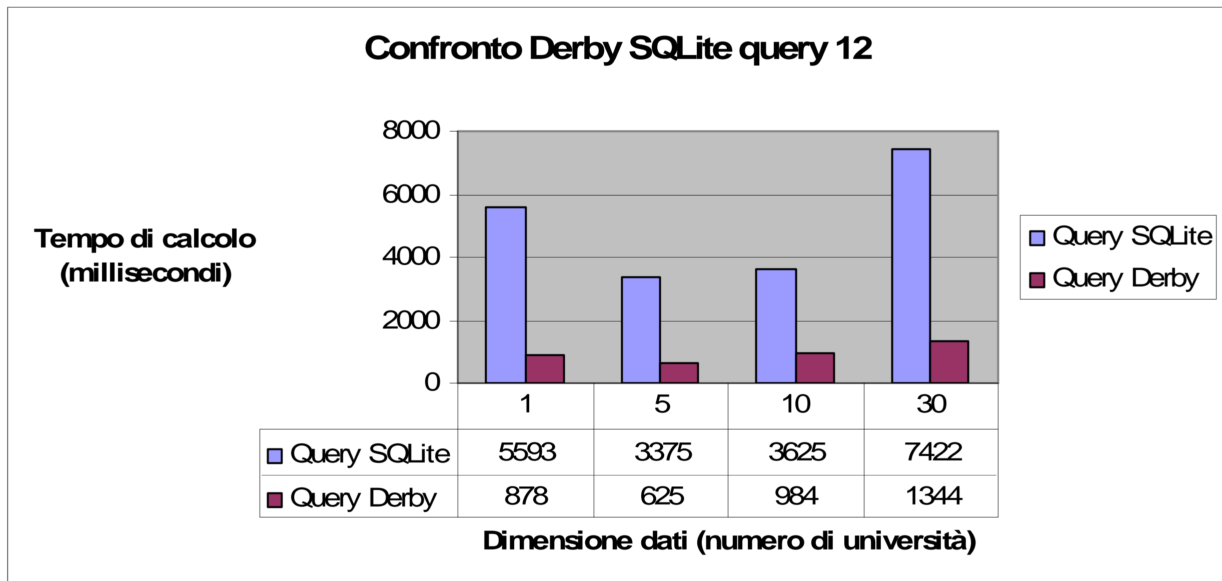
Confronto query 11



	1	5	10	30
Query SQLite	94	78	171	1797
Query Derby	94	406	797	916

La query 11 espressa in linguaggio del calcolo relazionale è:
{x | ResearchGroup(x) AND subOrganizationOf(x,"Univ0")}
Questa, come la query precedente, è una query semplice che presenta solo 3 query congiuntive nella query espansa. Questa query ha una particolarità, ossia la proprietà subOrganizationOf è definita transitiva. Ancora una volta, nonostante questa particolarità, si conferma il fatto che SQLite, sulla maggior parte delle query, risulta essere più veloce di Derby per Aboxes di piccole dimensioni, mentre per Aboxes di grandi dimensioni Derby riesce a fare answering in tempi più contenuti; questo discorso però non è valido nei casi in cui derby ha a che fare con i suoi punti critici come riportato nella discussione sulla query 6. Anche in questa query all'aumentare della dimensione della Abox i tempi si dilatano in modo approssimativamente lineare, ma solo per Derby, in quanto per SQLite tale discorso non si può applicare quando si passa da 10 a 30 università dove i tempi si dilatano di molto seguendo un andamento non proprio lineare.

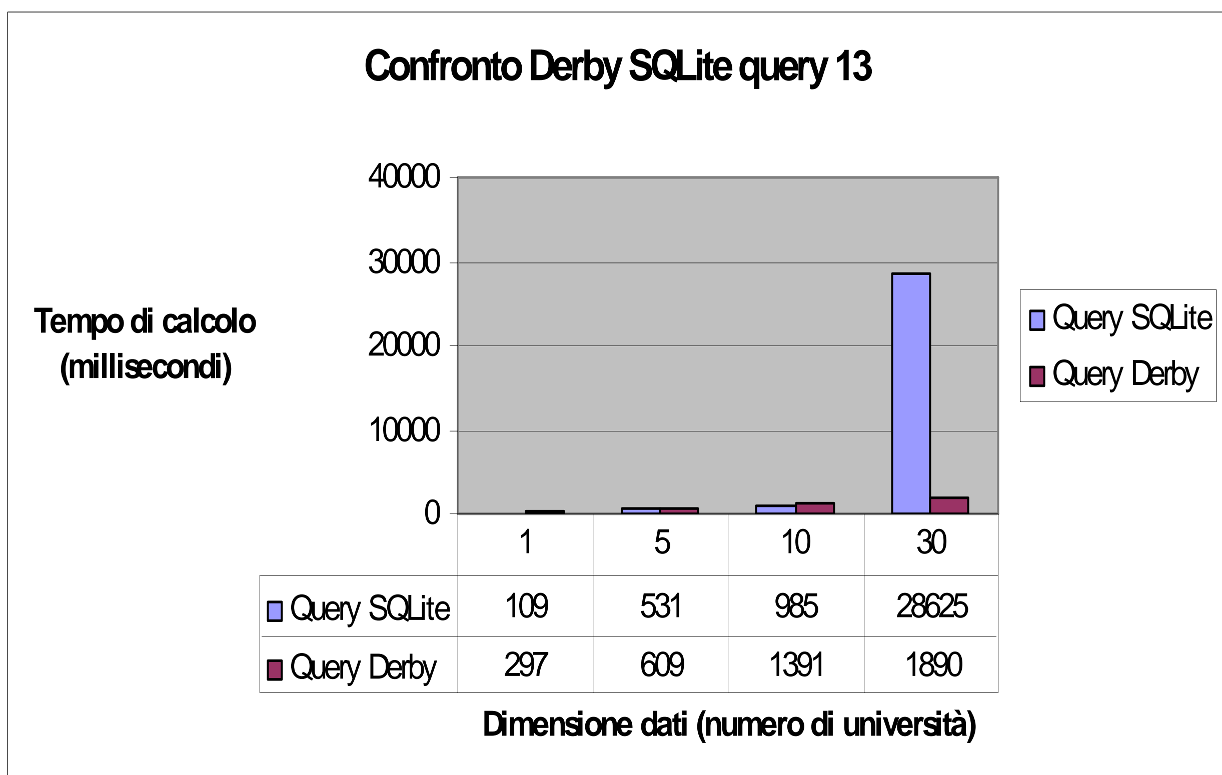
Confronto query 12



	1	5	10	30
Query SQLite	5593	3375	3625	7422
Query Derby	878	625	984	1344

La query 12 espressa in linguaggio del calcolo relazionale è:
{x,y | Chair(x) AND Department(y) AND worksFor(x,y) AND subOrganizationOf(y,"Univ0")}
Questa query nella union congiunctive query presenta 20 query congiuntive, ha una selettività molto forte, infatti in uscita presenta poche risposte. I risultati di questa query sembrano speculari rispetto ai risultati della query 4 dove Derby era molto più lento; ora ad essere molto più lento è SQLite. Anche questa volta, se prendiamo separatamente i risultati di Derby e SQLite, notiamo che per Abox relative ad 1 università i DBMS risultano leggermente più lenti rispetto a 5 università, però poi tra 5,10,30 università si nota il solito andamento crescente dei tempi di answering, questo probabilmente legato alle motivazioni discusse precedentemente per la query 8.

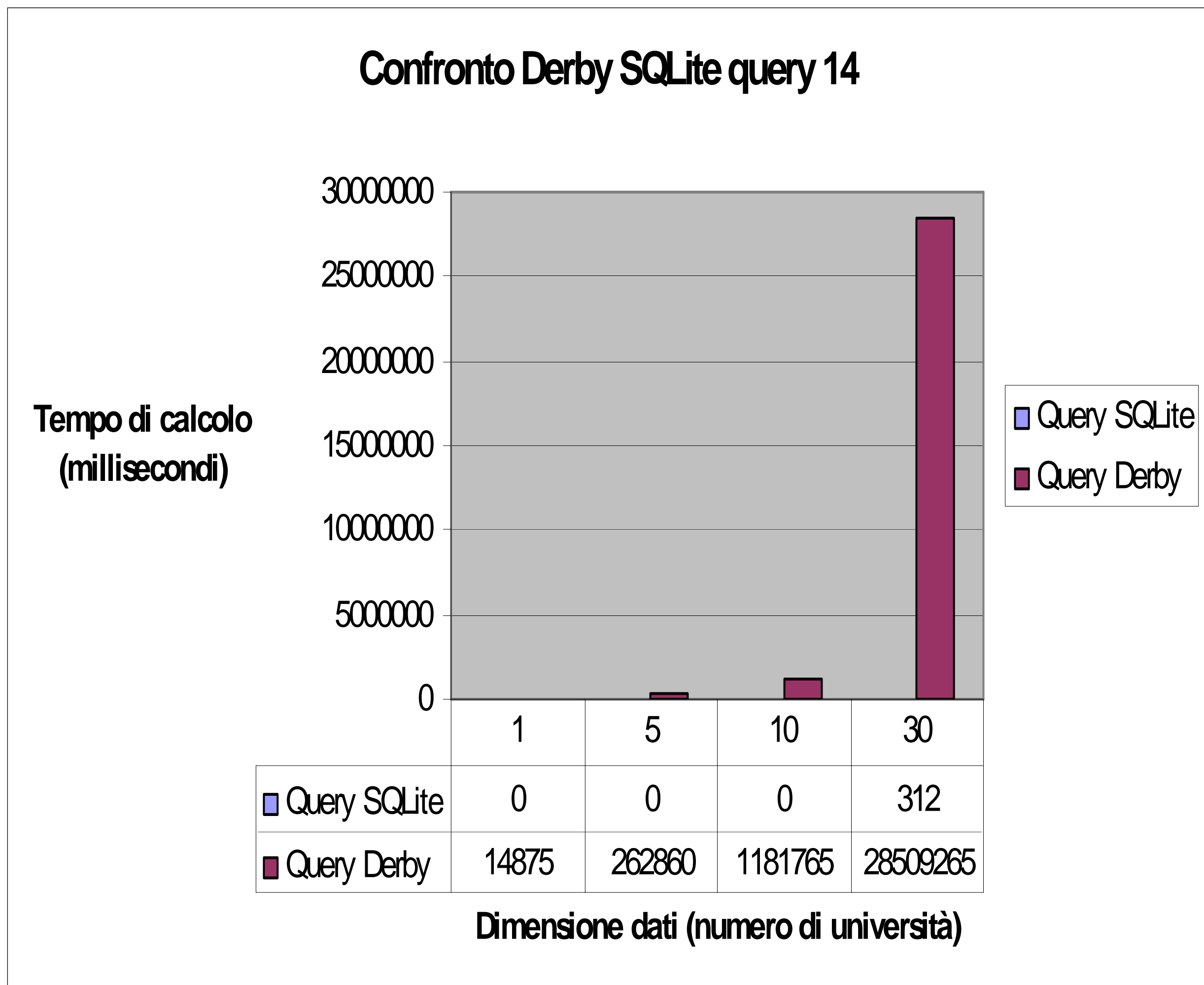
Confronto query 13



	1	5	10	30
Query SQLite	109	531	985	28625
Query Derby	297	609	1391	1890

La query 13 espressa in linguaggio del calcolo relazionale è:
{x | Person(x) AND hasAlumnus("Univ0",x)}.
QuOnto quando espande la query genera 10 conjunctive querys. Questa query serve per la verifica delle relazioni inverse. Come si può vedere dal grafico e dalle tabelle, per l'ennesima volta si conferma il fatto che SQLite, sulla maggior parte delle query, risulta essere più veloce di Derby per Aboxes di piccole dimensioni, mentre per Aboxes di grandi dimensioni Derby riesce a fare answering in tempi minori; e tale conferma è valida anche per query che verificano le relazioni inverse. L'andamento generalmente crescente per dimensioni via via maggiori delle Aboxes è ancora una volta confermato. Anche su questa query, nel passaggio da 10 a 30 università i tempi di answering di SQLite si dilatano tantissimo a conferma di quanto sopra affermato.

Confronto query 14



	1	5	10	30
Query SQLite	0	0	0	312
Query Derby	14875	262860	1181765	28509265

La query 14 espressa in linguaggio del calcolo relazionale è:
{x | UndergraduateStudent(x)}
Questa è la query più semplice, grande quantità in input, bassa selettività e assenza di gerarchia. I risultati del test di SQLite lo confermano, ma non possiamo certo dire la stessa cosa per Derby che in questa query come nella numero 6 mostra il suo grande lato negativo, ossia l'estrema lentezza nel fare answering sulle query di questo tipo, cioè query con grandi quantità di dati in ingresso e dove ci sono molti elementi da restituire visto la non selettività; infatti per l'Abox di 30 università le risposte trovate sono 269191. Il grafico parla da se, i tempi non sono neppure confrontabili, infatti per 30 università SQLite impiega 312 millisecondi, mentre Derby impiega 28509265 millisecondi, cioè quasi 8 ore.

Confronto query osservazione

Alcuni comportamenti risultati anomali, come una certa inversione di tendenza nella crescita dei tempi rispetto alla crescita della dimensione della Abox, su query molto semplici, quindi con tempi di calcolo molto contenuti (nella maggior parte dei casi), è probabile che siano dovuti a ritardi del computer durante l'esecuzione del test e non dipendenti dal DBMS utilizzato, questo perché si parla di pochi millisecondi, tempo in cui il computer potrebbe eseguire altri processi critici di sistema determinando un ritardo; tale fatto non può essere di giustificazione a differenze di ordine superiore (diversi secondi) che si incontrano in inversioni di tendenza di altre query (come ad esempio quanto avviene nella query 8).

Conclusioni

Dai risultati dei test possono essere tirate diverse conclusioni, prima fra tutte è che non esiste un DBMS tra SQLite e Derby che in ogni caso risulta essere migliore dell'altro, ognuno dei due ha pregi e difetti, ossia si comporta meglio su alcune query mentre su altre si comporta peggio, a seconda delle caratteristiche di queste ultime e ovviamente delle caratteristiche dei DB su cui vengono testate. Ora proveremo a delineare i casi in cui uno risulta essere meglio dell'altro.
Una caratteristica simile dimostrata tra Derby e SQLite è che in via generale (tranne alcuni casi particolari che sono già stati presi in considerazione in precedenza e che tra poco verranno ripresi) all'aumentare della dimensione della Abox i tempi di answering aumentano in modo pressoché lineare o quasi.
Un'altra caratteristica simile rilevata dai test riguarda la dilatazione dei tempi di calcolo per le query che riguardano un numero considerevole di Join, come si può vedere dai risultati dei test relativi alle query 2 e 9.
Dai test è risultato che nell'esecuzione di query con più di qualche attributo di concetto SQLite risulta più veloce rispetto a Derby indipendentemente dalla dimensione della Abox considerata.
Un altro risultato che dai test emerge in modo abbastanza chiaro è il fatto che in via generale Sqlite si comporta molto meglio per Aboxes di dimensioni ridotte, mentre per quanto riguarda Abox di grandi dimensioni (i tempi con SQLite si dilatano di molto), come quella considerata nei test da 30 università, salvo casi particolari come query con grandi quantità di dati in ingresso e dove ci sono molti elementi da restituire, Derby riesce a rispondere in tempi solitamente molto inferiori.
Dai test è uscito fuori un elemento importante, a volte sia Derby sia SQLite sull' Abox relativa ad una sola università impiegano per rispondere più tempo che su una Abox relativa a 5 università, questo perché probabilmente adotteranno delle ottimizzazioni solo per Abox da una determinata dimensione in poi, in quanto comunque sia, i tempi di risposta senza le supposte ottimizzazioni sono comunque accettabili.
Inoltre dai test emerge come già accennato un altro importante elemento, o meglio un vero e proprio limite di Derby, ossia l'estrema lentezza nel fare answering sulle query con grandi quantità di dati in ingresso e dove ci sono molti elementi da restituire (vedi querys 6 e 14).
Concludo ribadendo la migliore capacità di SQLite nel fare query answering su Abox di piccole dimensioni, mentre su grandi Abox è da preferire l'utilizzo di Derby; tuttavia se consideriamo nel complesso i test su SQLite e quelli su Derby possiamo dire che con SQLite, pur non avendo dimostrato di essere molto veloce su Aboxes grandi, non si sono verificate brutte sorprese con dei tempi inaccettabili come invece è avvenuto con Derby (query 6 e 14).

Appendice A (guida SQLite)

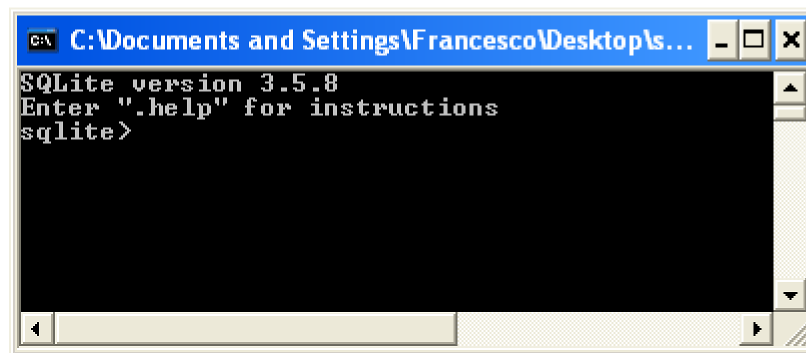
Uso di SQLite3

(Programma a linea di comando per la gestione di database basati su SQLite version 3)

1) Scaricare SQLite precompilato (ad es. per Windows il file da scaricare è: “sqlite-3_5_8.zip”) dall’indirizzo:
   a. http://www.sqlite.org/download.html
2) Scompattarlo in una directory a piacere

Esempio di come usare SQLite con “sqlite3.exe”

1) Avviare “sqlite3.exe”:



2) digitando il comando “.help” si ottiene la seguente schermata con la lista delle istruzioni eseguibili:

```
C:\Documents and Settings\Francesco\Desktop\sqlite3.exe
SQLite version 3.5.8
Enter ".help" for instructions
sqlite> .help
.bail ON|OFF           Stop after hitting an error.  Default OFF
.databases             List names and files of attached databases
.dump ?TABLE? ...      Dump the database in an SQL text format
.echo ON|OFF           Turn command echo on or off
.exit                  Exit this program
.explain ON|OFF        Turn output mode suitable for EXPLAIN on or off.
.header(s) ON|OFF      Turn display of headers on or off
.help                  Show this message
.import FILE TABLE     Import data from FILE into TABLE
.indices TABLE         Show names of all indices on TABLE
.load FILE ?ENTRY?     Load an extension library
.mode MODE ?TABLE?     Set output mode where MODE is one of:
                         csv      Comma-separated values
                         column   Left-aligned columns.  (See .width)
                         html     HTML <table> code
                         insert   SQL insert statements for TABLE
                         line     One value per line
                         list     Values delimited by .separator string
                         tabs     Tab-separated values
                         tcl      TCL list elements
.nullvalue STRING      Print STRING in place of NULL values
.output FILENAME       Send output to FILENAME
.output stdout         Send output to the screen
.prompt MAIN CONTINUE  Replace the standard prompts
.quit                  Exit this program
.read FILENAME         Execute SQL in FILENAME
.schema ?TABLE?        Show the CREATE statements
.separator STRING      Change separator used by output mode and .import
.show                  Show the current values for various settings
.tables ?PATTERN?      List names of tables matching a LIKE pattern
.timeout MS            Try opening locked tables for MS milliseconds
.width NUM NUM ...     Set column widths for "column" mode
sqlite> _
```

3) Come creare una tabella di nome ANAGRAFICA

a. Digitare come di seguito e premere "invio"

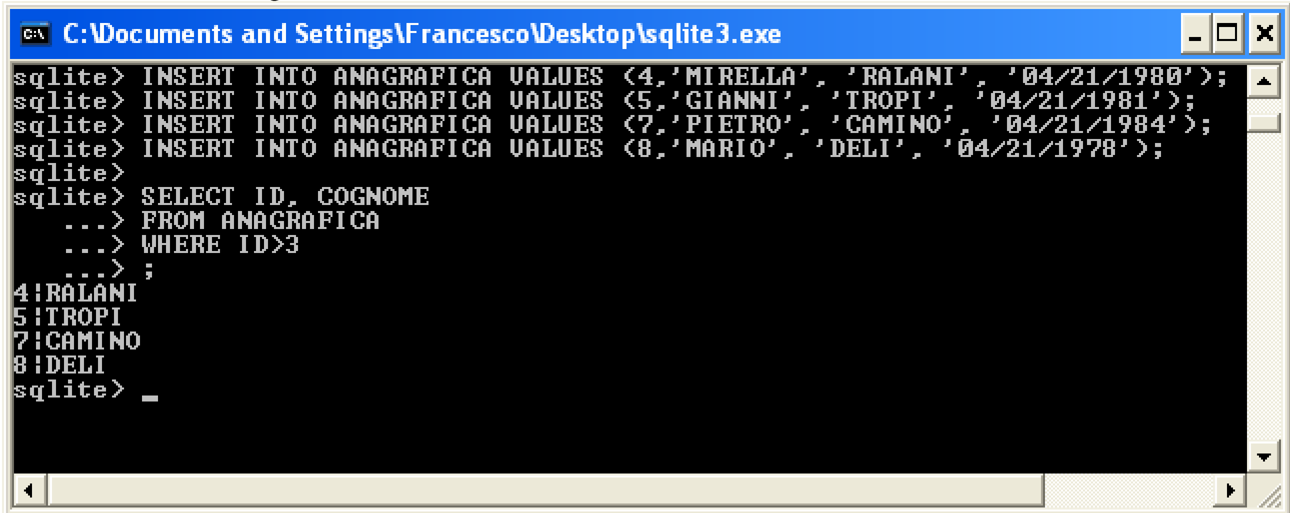
```
C:\Documents and Settings\Francesco\Desktop\...
SQLite version 3.5.8
Enter ".help" for instructions
sqlite> CREATE TABLE ANAGRAFICA
   ...>             (
   ...>              ID INTEGER PRIMARY KEY,
   ...>              NOME VARCHAR(25),
   ...>              COGNOME VARCHAR(25),
   ...>              DATA_NASCITA DATE
   ...>             );
```

4) Come inserire dati in una tabella

- Digitare come di seguito e premere "invio"

```
C:\Documents and Settings\Francesco\Desktop\sqlite3.exe
   ...>             COGNOME VARCHAR(25),
   ...>             DATA_NASCITA DATE
   ...>            );
sqlite> INSERT INTO ANAGRAFICA VALUES (1,'GIOVANNI', 'GIOVANNETTI', '11/01/1982'
);
sqlite> INSERT INTO ANAGRAFICA VALUES (2,'LUCA', 'RETTI', '02/24/1983');
sqlite> INSERT INTO ANAGRAFICA VALUES (3,'MARIO', 'PALUMBO', '04/21/1980');
sqlite> INSERT INTO ANAGRAFICA VALUES (4,'MIRELLA', 'RALANI', '04/21/1980');
sqlite> INSERT INTO ANAGRAFICA VALUES (5,'GIANNI', 'TROPI', '04/21/1981');
sqlite> INSERT INTO ANAGRAFICA VALUES (7,'PIETRO', 'CAMINO', '04/21/1984');
sqlite> INSERT INTO ANAGRAFICA VALUES (8,'MARIO', 'DELI', '04/21/1978');
sqlite> _
```

5) Come interrogare il database



6) Per maggiori dettagli sul funzionamento vedere:
   a. http://www.sqlite.org/sqlite.html

Utilizzo della Libreria SQLite

(In questo tutorial vedremo come utilizzare la libreria SQLite fornita da Zentus a cui si può accedere con il classico JDBC, e quindi il package java.sql)

7) Scaricare libreria SQLite:
   a. http://www.zentus.com/sqlitejdbc/index.html
8) Otterrete il file sqlitejdbc-vXXX-native.tgz, dove XXX indica la versione
9) Scompattarlo in una directory a piacere e otterrete 3 files es:
   a. README
   b. sqlitejdbc.dll
   c. sqlitejdbc-v044-native.jar
10) Nel caso in cui su Windows il file scaricato è un .tar, rinominatelo in .tgz, dopodichè scompattate il file ottenendo i tre files di sopra.
11) Prendete il file "sqlitejdbc-v044-native.jar" e inseritelo nella directory root della vostra applicazione
12) Prendete il file "sqlitejdbc.dll" e inseritelo nella directory bin della cartella java ad (es. C:\Programmi\Java\jre1.6.0_05\bin)

Esempio di come Integrare il DBMS nell'applicazione Java con SQLite

1) Prima di tutto bisogna procurarsi JDK
   a. Scaricare JDK (Java SE Development Kit (JDK)) dall'indirizzo: http://java.sun.com/javase/downloads/index.jsp
   o Installare la JDK seguendo le istruzioni
2) Creare (con un qualsiasi editor di testo) il file "Test.java" e metterlo in una directory a piacere (ad es. c:\prova) il cui codice esempio è il seguente:

```
import java.sql.*;

public class Test {
	public static void main(String[] args) throws ClassNotFoundException, SQLException {
		Class.forName("org.sqlite.JDBC");
		Connection conn = DriverManager.getConnection("jdbc:sqlite:test.db");
		Statement stat = conn.createStatement();
		stat.executeUpdate("create table people (nome, lavoro)");
		stat.executeUpdate("insert into people (nome, lavoro) VALUES ('Luca', 'studente')");
```

```
            stat.executeUpdate("insert into people (nome, lavoro) VALUES ('Andrea', 'modello')");
            stat.executeUpdate("insert into people (nome, lavoro) VALUES ('Michele', 'falegname')");
            ResultSet rs = stat.executeQuery("select * from people");
            while (rs.next()) {
                  System.out.println("Nome = " + rs.getString("nome"));
                  System.out.println("Lavoro = " + rs.getString("lavoro")+"\n");
            }
            rs.close();
            conn.close();
      }
}
```

3) Aprire un prompt dei comandi
4) Accedere alla directory dove precedentemente si è creato il file Test.java
5) Compilare da riga di comando digitando la seguente istruzione (N.B. "c:\sqlitejdbc-v044-native.jar" indica il percorso completo dove trovare "sqlitejdbc-v044-native.jar"):
   javac -classpath .;c:\sqlitejdbc-v044-native.jar *.java
6) Avviare il programma digitando la seguente istruzione(stessa nota di sopra):
   java -classpath .;c:\sqlitejdbc-v044-native.jar Test
7) Schermata che illustra tutti i passi precedenti con l'output dell'esecuzione del programma:

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Francesco>cd..

C:\Documents and Settings>cd..

C:\>cd prova

C:\prova>javac -classpath .;c:\sqlitejdbc-v044-native.jar *.java

C:\prova>java -classpath .;c:\sqlitejdbc-v044-native.jar Test
Nome = Luca
Lavoro = studente

Nome = Andrea
Lavoro = modello

Nome = Michele
Lavoro = falegname


C:\prova>
```

8) In pratica con tale codice si cerca la classe org.sqllite.JDBC e si crea una connessione al file database "test.db". Se il file non esiste viene creato. Dopodichè è possibile eseguire tutte le normali query SQL come si fa con qualsiasi DB tramite JDBC. Quindi l'applicazione così realizzata include il suo DBMS per gestire i dati dell'applicazione.

Appendice B (guida Derby)

Utilizzo della libreria Derby

(distribuzione lib, contiene solo i file .jar di Derby)

1) Scaricare Derby (lib distribution) dall'indirizzo:
   a. http://db.apache.org/derby/derby_downloads.html
2) Scompattarlo in una directory a piacere
3) Settare la variabile INSTALL_DERBY nella posizione in cui si è installato Derby, ad esempio:
   a. C:\> set DERBY_INSTAL=C:\derby

Esempio di come integrare il DBMS nell'applicazione java con Derby

1) Prima di tutto bisogna procurarsi JDK
   a. Scaricare JDK (Java SE Development Kit (JDK)) dall'indirizzo: http://java.sun.com/javase/downloads/index.jsp
   - Installare la JDK seguendo le istruzioni
2) Creare (con un qualsiasi editor di testo) il file "Test.java" e metterlo in una directory a piacere (ad es. c:\prova2) il cui codice esempio è il seguente:

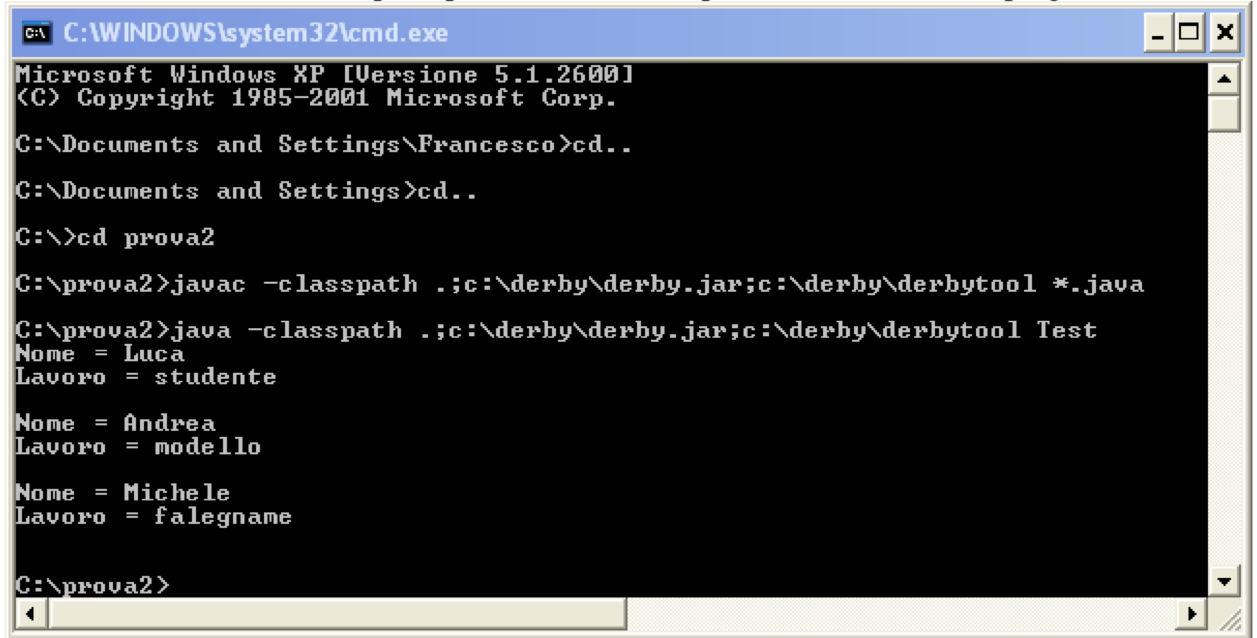
```
import java.sql.*;

public class Test {
        public static void main(String[] args) throws ClassNotFoundException, SQLException {

        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
        Connection conn = DriverManager.getConnection("jdbc:derby:prova;create=true");
        Statement stat = conn.createStatement();
        stat.executeUpdate("CREATE TABLE PERSONE (NOME VARCHAR(25), LAVORO
        VARCHAR(25))");
        stat.executeUpdate("INSERT INTO PERSONE VALUES ('Luca', 'studente')");
        stat.executeUpdate("INSERT INTO PERSONE VALUES ('Andrea', 'modello')");
        stat.executeUpdate("INSERT INTO PERSONE VALUES ('Michele', 'falegname')");
        ResultSet rs = stat.executeQuery("select * from PERSONE");
        while (rs.next()) {
                System.out.println("Nome = " + rs.getString("nome"));
                System.out.println("Lavoro = " + rs.getString("lavoro")+"\n");
                }
        rs.close();
        conn.close();
        }
}
```

3) Aprire un prompt dei comandi
4) Accedere alla directory dove precedentemente si è creato il file Test.java
5) Compilare da riga di comando digitando la seguente istruzione (N.B. "c:\derby\derby.jar" e "c:\derby\derbytool.jar" indicano il percorso completo dove trovare i files "derby.jar" e "derbytool.jar"):
   javac -classpath .;c:\derby\derby.jar;c:\derby\derbytool *.java
6) Avviare il programma digitando la seguente istruzione(stessa nota di sopra):
   java -classpath .; c:\derby\derby.jar;c:\derby\derbytool Test

7) Schermata che illustra tutti i passi precedenti con l'output dell'esecuzione del programma:



8) In pratica con tale codice si cerca la classe *org.apache.derby.jdbc.EmbeddedDriver* e si crea una connessione al file database "prova". Se il file non esiste viene creato. Dopodichè è possibile eseguire tutte le normali query SQL come si fa con qualsiasi DB tramite JDBC. Quindi l'applicazione così realizzata include il suo DBMS per gestire i dati dell'applicazione

9) Per maggiori dettagli visitare:
   a. http://db.apache.org/derby/manuals/index.html

Uso Client per Derby: SQuirreL SQL Client

Download JDK

1) Scaricare JDK (Java SE Development Kit (JDK)) dall'indirizzo:
   a. http://java.sun.com/javase/downloads/index.jsp
2) Installare la JDK seguendo le istruzioni

Download Derby

1) Scaricare Derby (lib distribution) dall'indirizzo:
   a. http://db.apache.org/derby/derby_downloads.html
2) Scompattarlo in una directory a piacere

Download e installazione SQuirreL SQL Client

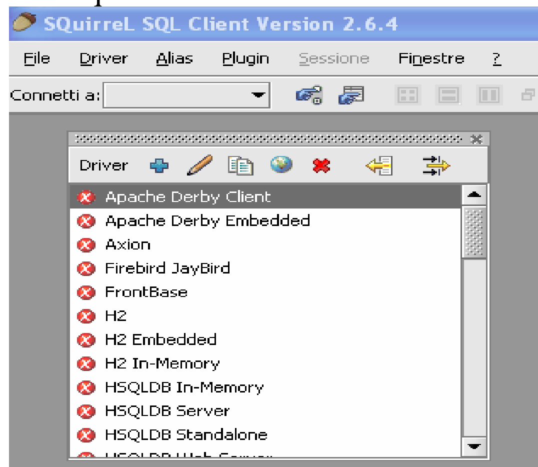
1) Scaricare SQuirreLSQL Client (installer) dall'indirizzo:
   a. http://squirrel-sql.sourceforge.net/
2) Lanciare l'installer scaricato
3) In alternativa al punto 2 eseguire il comando:
   - java -jar squirrel-sql-<version>-install.jar
   - es: java -jar squirrel-sql-2.6.4-install.jar
4) Seguire le istruzioni fornite dal programma di installazione
   - Quando apparirà questa schermata

- Spuntare:
  - Optional Plugin – Data Import
  - Optional Plugin – Derby
  - Optional Translation - <lingua preferita>
- Premere next

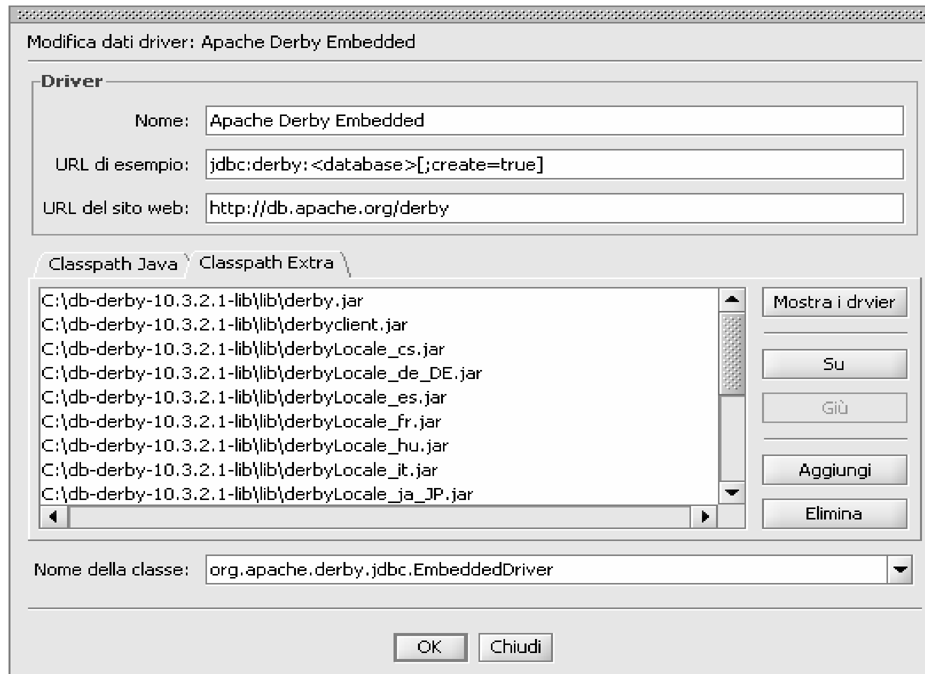
5) Attendere il completamento dell'installazione seguendo le istruzioni

Primo avvio e caricamento driver Derby

1) Avviare il programma e da questa schermata iniziale:



2) Selezionare "Apache Derby Embedded" (serve per creare un nuovo database) e ciccare sulla croce azzurra "Crea un nuovo driver"
   a. Aprire la scheda "Classpath Extra" e fare "Aggiungi"
   b. Posizionarsi nella cartella "lib" della cartella contenente i driver di Derby
   c. Selezionarli tutti, fare "Apri" e poi "Mostra driver". Dovreste vedere una simile schermata:

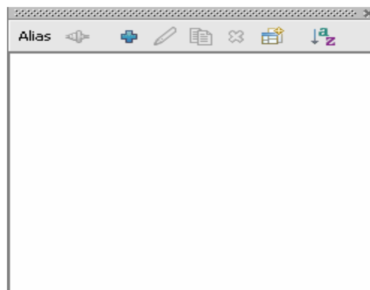


   d. Infine premere “Ok”

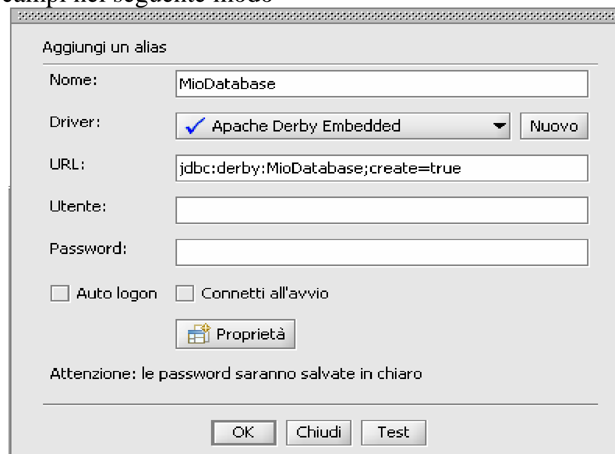
3) Ripetere il punto 2 per “Apache Derby Client” (serve per connettersi a database esistenti)

Creazione di un nuovo alias per un nuovo database

1) Da questa schermata:



2) Ciccare sulla croce azzurra “Crea nuovo Alias”
   a. Compilare i campi nel seguente modo

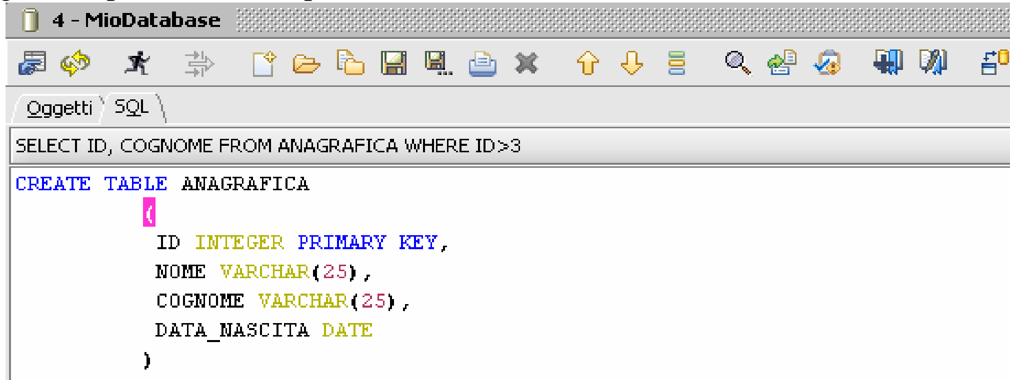


   b. Sostituire “Mio Database” con il nome che volete esso abbia
   c. Fare “Ok” poi “Connetti”

Creazione Tabella, inserimento dati e query

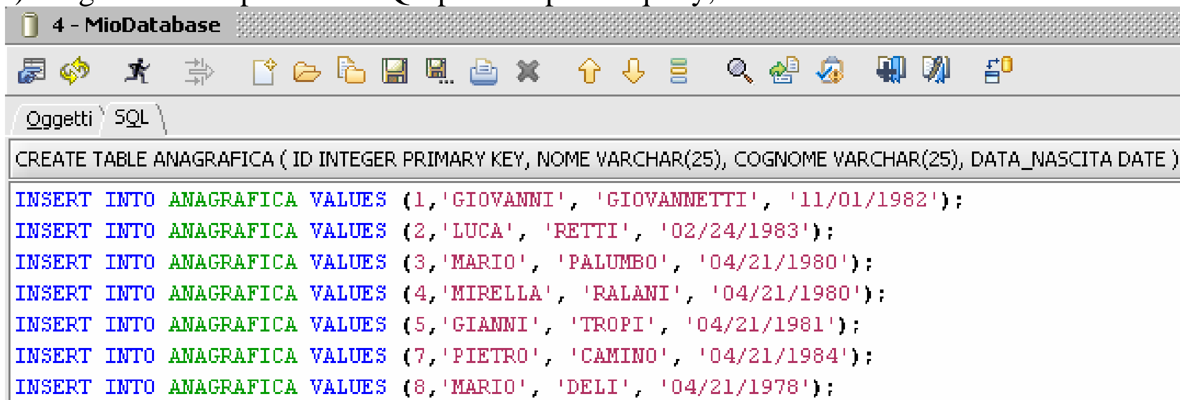
1) Connettersi al Database
2) Aprire la scheda “SQL”

3) Digitare espressione SQL per creare una tabella, ES:



a. Cliccare sull' "omino che corre" per processare il comando

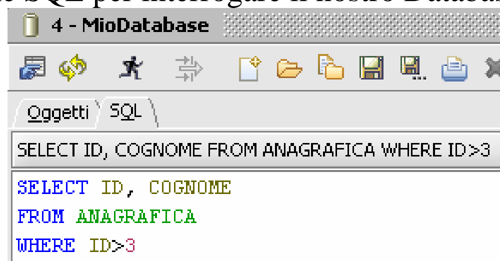
4) Digitare una espressione SQL per riempire la query, ES:



a. Cliccare sull'"omino che corre" per processare il comando

5) Per vedere la tabella con i dati inseriti:

6) Digitare un'espressione SQL per interrogare il nostro Database, ES:



a. Cliccare sull'"omino che corre" per processare il comando

7) Risultato Interrogazione

SELECT ID, COGN

SELECT ID, COGNOME FROM ANAGRAFICA WHERE ID>3

Estrazione | Metadata | Informazioni

ID	COGNOME
4	RALANI
5	TROPI
7	CAMINO
8	DELI