UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di laurea triennale in Ingegneria Informatica
TESI DI LAUREA

STUDIO DI UN ALGORITMO PER IL RICONOSCIMENTO DEL CONTENUTO DI ISTRUZIONI SQL

Laureando: Filippo Dal Pra'

Relatore: Dott. Giorgio Maria Di Nunzio

27 Settembre 2010

Anno Accademico 2009-2010

Ai miei fratelli,
Federico e Riccardo

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il progetto qui presentato è stato sviluppato nell'ambito di un tirocinio, della durata di 250 ore, svolto nel periodo maggio-luglio 2009 presso l'azienda ASI srl. L'azienda, con sede a Padova, è una moderna società presente nel mercato dell'Information Technology da oltre 20 anni, con attività di progettazione e sviluppo di prodotti software di tipo industrializzato e di consulenza applicativa ed organizzativa; essa è Business Partner IBM, Gold Certified Partner di Microsoft e da anni collabora con la Facoltà di Scienze dell'Università di Padova su progetti di ricerca e sviluppo che interessano le tecnologie ed i prodotti su cui opera.
Impegnata per la maggior parte nell'Italia settentrionale, ASI srl è un'azienda che conta su una clientela di circa 250 aziende, operanti in vari settori, dal manifatturiero al commerciale; questa alta diversificazione della clientela porta l'azienda allo sviluppo di molteplici prodotti, ognuno dei quali meglio si adatta alle richieste delle imprese.

La suite applicativa plain del gruppo ASI fornisce un insieme software aziendali in grado di meglio realizzare le richieste dei loro clienti, tra cui applicativi per la gestione del patrimonio documentale, della logistica, delle risorse umane, del magazzino, della produzione e molto altro. Tra tutti questi software, plain extended data publisher permette all'utente una facile ed immediata interazione con il database aziendale o con strutture di memorizzazione esterne: dotato di un'interfaccia grafica intuitiva, riesce a reperire velocemente le informazioni necessarie all'utente attraverso la creazione semplificata di query, elaborandole e convertendole poi in un formato leggibile e coerente.
I risultati delle query richieste vengono visualizzati sia in formato grafico che in formato tabellare, mantenendo sempre il codice sorgente SQL con cui si è creata la query; ciò da la possibilità di modificare o copiare la query per necessità future.
Il software, riuscendo a gestire correttamente gli input dati dall'utente in fase di

creazione, riesce a passare correttamente dalla creazione manuale di una query al suo risultato grafico; dal risultato grafico è inoltre possibile ricavare lo script SQL della query.
L'obiettivo dell'azienda era quello di voler dare all'utente la possibilità, dato lo script SQL ottenuto dalla query rappresentata graficamente, di poterla nuovamente visualizzare in ambiente grafico.
La focalizzazione su tale richiesta è stata l'attività su cui si è concentrato il tirocinio: individuare ed implementare un algoritmo che riuscisse, data una query in codice SQL, ad analizzarla ed a riconoscerne tutti gli elementi costitutivi, in modo che essi poi potessero essere processati nel modo corretto dal software, permettendone la ricostruzione a livello grafico senza doverla ricreare da zero.
Il progetto ha avuto inizio con una fase di formazione sulle filosofie di riconoscimento di una stringa e su un accurato studio della sintassi SQL standard 1992, al fine di interpretare correttamente ogni singolo componente del codice. In seguito si è redatto un elenco di tutte le caratteristiche che l'algoritmo avrebbe dovuto saper riconoscere in fase di elaborazione all'interno del codice ed infine, utilizzando WPF (Windows Presentation Foundation) e gli oggetti a disposizione del framework, è stato realizzato e testato l'algoritmo.
L'approccio effettuato per la scomposizione e l'analisi del codice SQL di una query è stato effettuato per livelli: partendo dalla query generale, si è effettuata una suddivisione nelle sue clausole più importanti (SELECT, FORM, WHERE, GROUP BY, HAVING e ORDER BY). In seguito, ogni clausola è stata suddivisa in spezzoni contenenti i suoi costrutti fondamentali (ad esempio i campi nella clausola SELECT o i blocchi di condizioni nella clausola WHERE) ed infine, all'ultimo livello di analisi, per ogni spezzone sono state individuate le parole chiave, gli operatori e gli elementi ivi contenuti.
Poiché l'analisi della stringa SQL deve essere effettuato in modo preciso, è fondamentale che ogni elemento della stringa venga riconosciuto ed interpretato in maniera corretta per consentire l'esatta ricostruzione della query e delle relazioni che implica. Infatti ogni elemento, in relazione al contesto in cui è inserito, verrà valutato in modo differente e sarà valorizzato in modo particolare. Per questo uno dei requisiti fondamentali è che le query che vengono inserite per l'analisi siano sintatticamente e logicamente strutturate in modo corretto al fine di prevenire eventuali errori del programma. Dopo averne effettuato l'analisi,verrà creato un oggetto di tipo Query, definito nella libreria plain.Portal.DDPv2, a cui verranno associate tutte le caratteristiche e proprietà rilevate dalla scomposizione della query stessa; tale oggetto verrà poi specificato in un file XML (eXtensible Markup Language).

L'oggetto di tipo Query creato possiede tutte le informazioni necessarie e valorizzate in modo tale da permettere alla suite aziendale di poterlo utilizzare come se fosse un oggetto proprio: si riesce cosi a raggiungere lo scopo principale del progetto, ovvero la ricostruzione grafica di una stringa SQL.
Nella fase finale dell'elaborazione, sono state prodotte e proposte all'azienda due versioni dello stesso progetto. Entrambe realizzate allo stesso modo, esse differiscono solo nella tipologia di utilizzo: QueryAnalyzer è una applicazione via console, mentre LibQueAnalyzer è una libreria di classi.
La versione applicativa, che ha avuto rilevanza principale nella stesura del progetto, è strutturata in modo tale da poter ricevere in ingresso, sia via console che tramite lettura da un file, una query in forma di codice SQL, che verrà poi scomposta ed analizzata secondo l'idea sopra esposta. LibQueAnalyzer, funzionante come QueryAnalyzer, differisce da essa soprattutto per la peculiarità di essere una libreria di classi: questa scelta è stata fatta per poterne consentire un più corretto inserimento all'interno della suite aziendale plain Extended Data Publisher. La libreria tratta in ingresso la stringa SQL da valutare, e restituisce un oggetto di tipo Query che verrà utilizzato per ricreare graficamente la query analizzata. Inoltre, per meglio adattarsi al contesto in cui è stata inserita, alcuni metodi sono stati modificati al fine di memorizzare alcune proprietà essenziali che altrimenti andrebbero perse con l'elaborazione.

Si passeranno qui in rassegna tutte le fasi del progetto, analizzando passo dopo passo tutte le problematiche e le soluzioni adottate nello sviluppo dell'algoritmo. Inoltre, vista la possibilità ed il tempo avuti a disposizione, sono state implementate funzionalità aggiuntive alle richieste dell'azienda che verranno citate e discusse; infine, verranno presentati i risultati ottenuti.

Desidero ringraziare coloro che mi hanno dato l'opportunità di svolgere questa importante esperienza: il Sig. Carlo Venturella, Responsabile Progettazione e Sviluppo Software; il Sig. Marco Alquati, del servizio Produzione Software plain portal-documentale, per l'aiuto e la disponibilità accordatami nei tre mesi di tirocinio.

Voglio esprimere la mia riconoscenza al Prof. Giorgio Maria Di Nunzio per avermi dato la sua fiducia e disponibilità per lo sviluppo di questa tesi.

Capitolo 2

Database e Linguaggio

2.1 Il Database

Oggigiorno l'evoluzione della tecnologia informatica consente una ricerca più rapida ed efficace di qualsiasi tipo di informazione ed in qualsiasi postazione ci si trovi. L'innovazione più importante per questo tipo di evoluzione ci è offerta dal database, un insieme di dati suddivisi per categorie secondo uno schema logico, le tabelle, e poi ordinati per caratteristiche, i campi. Ogni informazione contenuta all'interno del database viene quindi inserita in una particolare tabella e caratterizzata da valori specificati dai vari campi: tale informazione viene chiamata record. Oltre ai record, le basi di dati contengono anche altre informazioni a livello gestionale, quali le modalità di rappresentazione dei dati, i permessi di accesso e le relazioni che legano tra loro le informazioni ivi contenute.

I database vengono creati, gestiti ed interrogati da un sistema software chiamato DataBase Managment System, o DMBS: è un sistema di gestione dei dati che garantisce un livello di sicurezza ai dati, permettendone una condivisione sicura ed affidabile agli utenti.
Tale sistema si interpone tra l'utente e i dati nel database; grazie a questo strato di software l'utente non ha accesso diretto ai dati memorizzati fisicamente, ma solo ad una loro rappresentazione logica, permettendo cosi un alto livello di indipendenza tra dati fisici ed applicazioni. Le attuali applicazioni dei database permettono l'accesso ai dati a più utenti contemporaneamente: questo è reso possibile dal fatto che vi sono DBMS sviluppati in modo che, utilizzando una sola copia dei dati, permettono la creazione di più rappresentazioni logiche di questi, riducendone la ridondanza e l'inconsistenza.
Inoltre i DBMS devono gestire il sistema di permessi, così ciascun utente a seconda delle autorizzazioni potrà leggere, scrivere, modificare o eliminare dati.

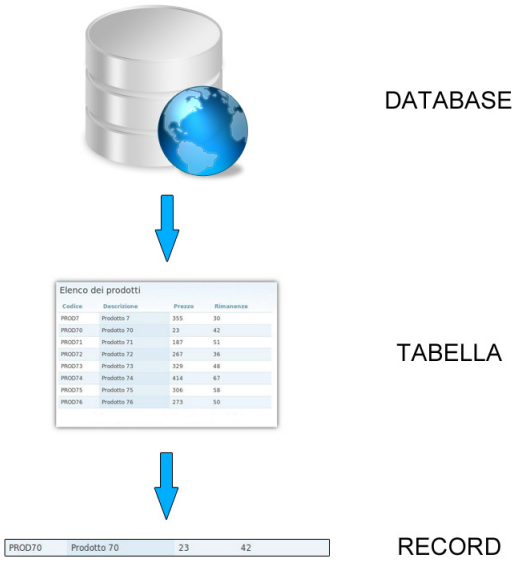

Figura 2.1: Elementi di un database

Vi sono diversi modi con cui un DBMS attua la sicurezza dei dati:

- Partizionando il database;
- Cifrando i dati contenuti nel database;
- Marcando i dati con etichette che ne definiscono la sensibilità ed i permessi di accesso;
- Creando modalità di accesso individuali al database tramite credenziali verificate;
- Mostrando all'utente delle viste, ovvero un sottoinsieme del database che contiene le informazioni a cui l'utente può avere accesso.

Svolgono un ruolo fondamentale in numerose applicazioni informatiche, dalla contabilità, alla gestione delle risorse umane e alla finanza fino a contesti tecnici come la gestione di rete o la telefonia.
Vi sono diversi tipi di organizzazione per un DBMS: il modello gerarchico in cui i dati sono organizzati in record connessi tra loro secondo strutture ad albero, il modello reticolare in cui i record sono legati tra loro con strutture ad anello che permettono all'utente finale di accedere ai dati in maniera più semplice, e

quello oggi più diffuso, il modello relazionale: l'assunto fondamentale è che tutti i dati sono rappresentati come relazioni, e manipolati con gli operatori dell'algebra relazionale. Tale modello consente al progettista di database di creare una rappresentazione consistente e logica dell'informazione: le tabelle non sono altro che delle relazioni, sistemi che legano tra loro determinati record a seconda di determinate caratteristiche in comune. Per meglio comprenderle le tabelle sono suddivise in campi: ognuno dei campi può assumere valori specifici di un tipo ben definito ed all'interno di un dominio limitato; sono proprio questi valori a mettere in relazione i record tra loro.
Dunque un database relazionale è un insieme di relazioni contenenti valori e il risultato di qualunque interrogazione o manipolazione dei dati può essere rappresentato anch'esso da relazioni.

Il database quindi deve essere progettato e costruito per consentire l'accesso, tramite i DBMS, alle informazioni in esso contenute ed alla sua struttura da parte degli utenti: per questo sono stati sviluppati diversi tipi di linguaggi, a seconda del loro utilizzo:

- Data Definition Language (DDL): consente di definire la struttura della base di dati e le autorizzazioni per l'accesso;
- Device Media Control Language (DMCL): permette alla struttura fisica del database di far riferimento alle particolari unità di memoria di massa utilizzate dal sistema;
- Data Manipulation Language (DML): consente di interrogare e aggiornare le istanze della base di dati;
- Data Control Language (DCL): permette la gestione dell'accesso al database con relative restrizioni di operazioni come aggiornamento, selezione e cancellazione;
- Query language (QL): permette di interrogare il database al fine di ritrovare i dati relativi alla chiave di ricerca impostata dall'utente.

Il linguaggio più diffuso e basilare per interagire con un database è il linguaggio SQL.

2.2 Il linguaggio SQL

L'SQL (Structured Query Language) è un linguaggio di tipo dichiarativo usato in sistemi basati sul modello relazionale per leggere, modificare e gestire dati memorizzati nei database, per creare e modificare schemi di database, per creare e gestire strumenti di controllo ed accesso ai dati.

L'SQL nasce nel 1974 nei laboratori dell'IBM [1] come strumento per interagire con i database. I primi sviluppi e sperimentazioni di questo progetto portarono, nel 1977, a una nuova versione del linguaggio, inizialmente chiamata SEQUEL/2 ed in seguito ribattezzata SQL per motivi legali. Su di esso si sviluppò il prototipo System R [11], che venne utilizzato da IBM per usi interni e per alcuni suoi clienti. Dato il suo successo riscontrato, anche altre società iniziarono subito a sviluppare prodotti basati su SQL. Nel 1981 IBM iniziò a vendere alcuni prodotti relazionali e nel 1983 rilasciò DB2 [2], il suo DBMS relazionale diffuso ancor oggi. SQL divenne subito lo standard industriale per i software che utilizzano il modello relazionale: l'ANSI lo adottò come standard fin dal 1986, senza apportare modifiche sostanziali alla versione inizialmente sviluppata da IBM, e nel 1987 la ISO fece lo stesso. Questa prima versione standard è denominata SQL/86. Negli anni successivi si realizzarono altre versioni, che furono SQL/89, SQL/92 e SQL/2003. Tale processo di standardizzazione mirava alla creazione di un linguaggio che funzionasse su tutti i DBMS (Data Base Management Systems) relazionali, ma questo obiettivo non fu raggiunto. Infatti, i vari produttori implementarono il linguaggio con numerose variazioni e, in pratica, adottarono gli standard ad un livello non superiore al minimo, definito dall'Ansi come Entry Level.
Nel corso degli anni SQL si è evoluto con l'introduzione di costrutti procedurali, istruzioni per il controllo di flusso, tipi di dati definiti dall'utente e varie altre estensioni non originariamente progettati.

Le funzioni principali del linguaggio SQL sono quello di interrogare, aggiornare ed inserire dati all'interno di un database; tali operazioni, basilari del linguaggio SQL, vengono effettuate attraverso l'uso di particolari istruzioni, denominate query e Data Manipulation Language (DML).
Queste operazioni permettono all'utente di descrivere quali sono le informazioni a lui necessarie, lasciando al DBMS la responsabilità di amministrarle, disporle e effettuare le operazioni fisiche per produrne visivamente i risultati. Sono righe di comando, in codice SQL, che descrivono al DBMS le operazioni da effettuare:

[1] http://www.ibm.com/
[2] http://www-01.ibm.com/software/data/db2/

vengono definite attraverso l'uso di parole chiave che ne indicano le istruzioni da eseguire, e da un elenco di attributi che indica su quali tabelle, campi o record sono da effettuare le operazioni.
Mentre le istruzioni DML possono modificare la struttura del database, le query servono per richiedere un set particolare di informazioni; inoltre le query possono essere collegate tra loro grazie a particolari operatori insiemistici, atti a confrontare risultati di più richieste, oppure possono essere utilizzate per formulare tabelle necessarie ad un'altra query. Tutte queste possibilità aiutano l'utente ad avere risultati più specifici per il suo interesse, ma sono accompagnate da una maggiore complessità nella formulazione. In questo capitolo si vedranno i principali costrutti delle istruzioni più comuni, lasciando per ultime le query, più interessanti, che verranno approfondite in dettaglio.

2.3 Istruzioni di modifica, eliminazione e creazione

I comandi SQL di modifica, eliminazione e creazione possono interagire con la struttura del database: esse, accedendo ai dati, possono modificare, creare o cancellare tabelle, campi e record. Sono delle operazioni delicate, che possono compromettere l'integrità delle informazioni contenute nella base dati. Per questo, per determinati elementi, vengono creati dei permessi affinché solo chi ne è in possesso possa operare su di essi; in questo modo si previene la modifica al database da parte di utenti non autorizzati, lasciandone la responsabilità solo a chi di competenza.
Le parole chiave sono molteplici per queste operazioni, di seguito verranno elencate le più comuni:

- UPDATE: ha la funzione di modificare i dati delle tabelle; può modificare, a seconda della sintassi usata, un solo campo, più campi contemporaneamente, tutti i campi oppure solo una determinata selezione dei campi utilizzando il comando WHERE;

- DELETE: ha la funzione di cancellare i dati dalle tabelle; come il comando update anche delete può operare in modo generico cancellando tutti i record della tabella, oppure può identificare i record da cancellare mediante la parola chiave aggiuntiva WHERE e la condizione (o le condizioni) ad essa associata;

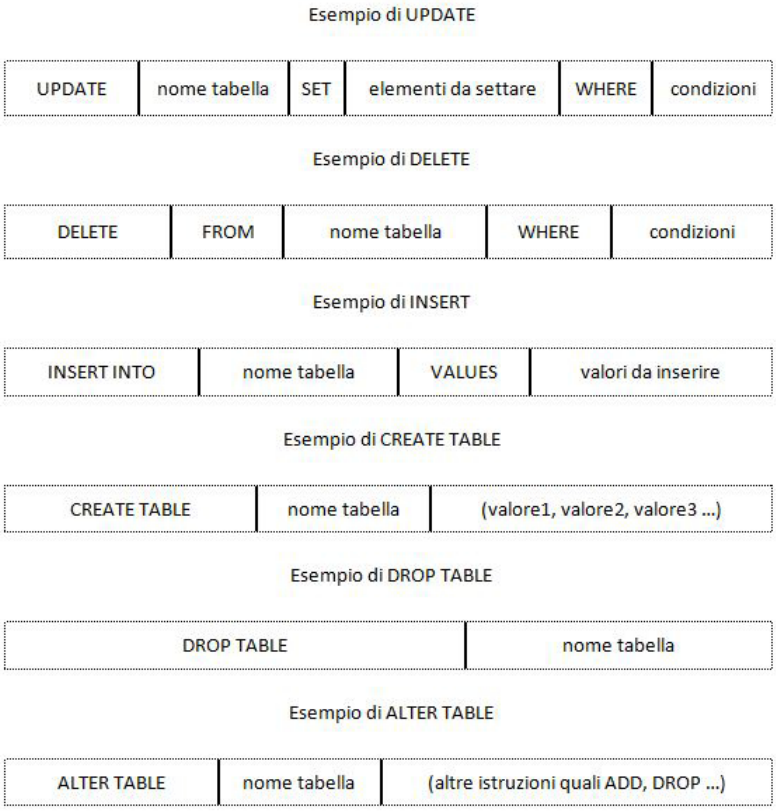

Figura 2.2: Esempi di comandi SQL

- INSERT: ha la funzione di inserire i dati nelle tabelle;
- CREATE TABLE: ha la funzione di creare tabelle all'interno della base dati; dopo il comando si possono specificare, record per record, i campi e i relativi valori da associarvi;
- DROP TABLE: ha la funzione di eliminare una tabella;
- ALTER TABLE: ha la funzione di modificare i dati contenuti in una tabella.

Esempi della sintassi di tali comandi sono mostrate in figura 2.2.
Non va dimenticato che, in alcuni casi, una tabella coinvolta per uno di questi comandi potrebbe essere una tabella ottenuta da una query di tipo interrogativa, o una query annidata.

2.4 Le query

La query è il costrutto più importante e più usato del linguaggio SQL: è una funzione che permettere di interrogare il database in modo da ottenere delle informazioni ivi contenute filtrandole, a seconda delle esigenze, con l'introduzione di speciali costrutti e parole chiave. Le query sono infatti le stringhe con il più ricco insieme di parole chiave utilizzabili: questo porta alla possibilità di un più alto grado di specificità nella formulazione delle richieste, a dettagli ben più specifici ma come controparte ad una maggiore complessità sintattica. La lunghezza delle query infatti può arrivare anche a più di 70 righe di codice.
La sintassi di una query è ben definita a priori, con regole precise che ne identificano ogni singola componente; le due parole chiave obbligatorie per la formulazione di una query sono:

- SELECT: indica i campi che si intendono ricercare nelle tabelle specificate e ne evidenziano il risultato;
- FROM: indica le tabelle in cui l'utente intende ricercare i valori dei campi richiesti.

Questo semplice costrutto, sebbene poco articolato, è già in grado di produrre un risultato.
Nel linguaggio SQL sono state implementate altre parole chiave per le query che, seppur facoltative, ne aumentano la potenza espressiva, e per questo sono frequentemente utilizzate:

- WHERE: questa parola chiave indica delle condizioni che devono essere verificate nell'interrogazione al database: infatti verranno visualizzate nel risultato solo ed esclusivamente i record che abbiano verificato le condizioni qui espresse. L'omissione di questa clausola indica che la query selezionerà tutti i record presenti nel risultato del prodotto cartesiano delle tabelle nella clausola FROM;
- GROUP BY: i record risultanti dalla query verranno suddivisi in gruppi a seconda del valore del campo qui inserito, chiamato campo di raggruppamento;
- HAVING: usata solo in presenza della clausola GROUP BY, viene utilizzata per specificare delle condizioni su ciascun gruppo, corrispondente ad ogni distinto valore del campo di raggruppamento: solo i gruppi per i quali la condizione è soddisfatta entrano a far parte del risultato della query;

- ORDER BY: questa parola chiave identifica quali campi vengono utilizzati per ordinare i risultati ed in che senso vengono ordinati (crescente o decrescente). Senza questa specifica, il risultato della query è semplicemente non definito.

Il risultato di una query è quindi una tabella costituita da record:

- che appartengono alle tabelle elencate nella clausola FROM;
- che soddisfano le condizioni espresse della clausola WHERE (ove presente);
- che sono proiettate sulla lista di attributi specificati nella clausola SELECT;
- raggruppati secondo il campo espresso in GROUP BY (ove presente);
- rispettanti la condizione di raggruppamento inclusa in HAVING (ove presente);
- ordinati secondo le specifiche di ORDER BY (ove presente).

Va posta particolare attenzione al fatto che alcuni campi possono presentare un valore nullo: in tal caso può significare una valore non applicabile, un valore applicabile ma non conosciuto oppure una di queste due condizioni ma non si sa quale. Lo standard SQL prevede una logica a tre valori per la soluzione di operazioni tra valori nulli: vero, falso e sconosciuto.

In generale, le clausole presenti in maggior parte nelle query sono SELECT, FROM e WHERE; l'omissione di quest'ultima non crea problemi nella correttezza della query, ma porta il sistema a fornire un risultato corposo, privo di ogni logica e dispendioso in termini di tempo di calcolo.

Ognuna delle clausole sopra elencate e i loro successivi attributi, formano una parte importante nella formulazione di una query. Non solo, ognuna di esse al suo interno è dotata di altre parole chiave che apportano altre condizioni ed operazioni interne alle specifiche e concorrenti nella formazione del risultato. Questi altri costrutti, importanti ai fini della nostra analisi, verranno ampiamente discussi in seguito.

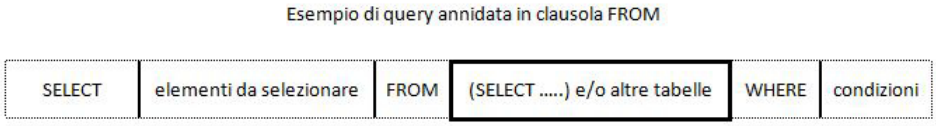

Figura 2.3: Esempio di query annidata in clausola FROM

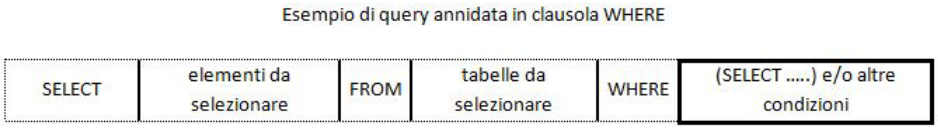

Figura 2.4: Esempio di query Annidata in clausola WHERE

2.5 Le query annidate

Come abbiamo già accennato, il risultato di una query è una tabella che rispetta le specifiche inserite dall'utente nelle varie clausole accessorie. Tale tabella non ha nessuna differenza con una qualsiasi altra tabella nel database, se non che per il contenuto. Per questo nulla vieta di utilizzare una query all'interno di un'altra query.
Le query utilizzate come valore all'interno di clausole di un'altra query vengono chiamate query annidate (o nidificate). Non devono essere confuse con le query legate dagli operatori insiemistici, poiché quelle rappresentano due query diverse; in questo caso, si ha che il codice di una query è inserito all'interno di una clausola di un'altra query.
La query annidata può essere inserita in una clausola FROM, e quindi valutata come tabella, oppure in una clausola WHERE, dove ne vengono utilizzati gli attributi per creare delle condizioni. In ogni caso, all'utente finale verrà presentato il risultato globale della query, ma non quello della query annidata, in quando essa viene processata come un parametro operativo.
Questo tipo di costrutto, sebbene piuttosto complesso, permette delle operazioni più complesse eliminandone i passaggi intermedi. Richiedono tuttavia particolare attenzione in fase di costruzione.

Un esempio di struttura di una query annidata posta all'interno della clausola FROM è mostrato in figura 2.3, mentre per una query annidata inserita tra le condizioni della clausola WHERE si ha il costrutto di figura 2.4. In questi due

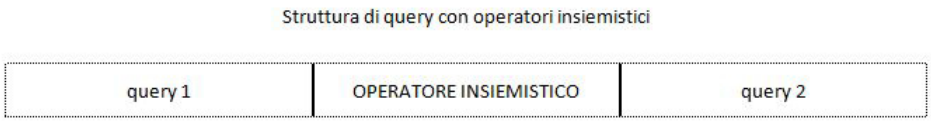

Figura 2.5: Struttura di query con operatori insiemistici

esempi si nota una delle differenze fondamentali tra le query annidate a seconda della clausola in cui sono inserite: nel primo esempio la query annidata viene presa e trattata come una tabella, nel secondo caso, essendo all'interno di un blocco condizionale, la sua funzione sarà quella di restituire dei valori confrontabili con altri per verificare la condizione richiesta.

2.6 Gli operatori insiemistici

In alcune occasioni l'utente può voler utilizzare dei risultati derivanti dal confronto o dall'unione di più tabelle: questo richiede l'utilizzo di due query diverse e l'utilizzo di un operatore che le colleghi tra loro, specificando la funzione da applicare. Tale insieme di operatori vengono detti insiemistici.
Questi operatori sono di facile implementazione, e sebbene richiedano la scrittura di due query il risultato finale è di grande interesse; l'unico importante requisito affinché il risultato sia corretto è che i dati che si richiede vengano confrontati siano tra loro compatibili. La struttura globale per questi operatori è mostrata in figura 2.5

Questi operatori vanno inseriti al livello esterno del codice,ovvero alla fine della prima query e prima dell'inizio della seconda e, a seconda del tipo di operatore, si possono avere i seguenti risultati:

- UNION: calcola il risultato dell'unione di due tabelle, eliminando eventuali risultati duplicati;
- EXCEPT o MINUS: calcola il risultato dell'unione di due tabelle evidenziando i valori che rispettano i vincoli esclusivi della prima query e non della seconda;
- INTERSECT: calcola il risultato dell'unione di due tabelle evidenziando i valori che rispettano i vincoli esclusivi di entrambe le query interrogative.

Tutti questi operatori non tengono conto di risultati duplicati, e per questo li eliminano; volendo, è possibile conservare i risultati duplicati aggiungendo al termine degli operatori sopra citati la parola chiave ALL.

2.7 Analisi della sintassi

Il codice SQL è regolato da una sintassi chiara e ben definita. Per poter eseguire delle query, essa richiede che tutte le specifiche necessarie all'esecuzione del comando vengano rispettate.
Come abbiamo visto, una query può essere composta da più o meno clausole a seconda delle necessità, ma tutte devono seguire rigidi vincoli costruttivi, pena l'esecuzione errata della richiesta.
Il punto di forza nello studio dell'algoritmo qui presentato è rappresentato proprio dalla sintassi: seguendo uno schema ben definito, si ha la possibilità di ricercare all'interno della stringa le clausole note per le query e, in modo via via più particolareggiato, i campi, le condizioni e le tabelle che ogni clausola dichiara.
Ogni clausola poi ha una sua sintassi, anch'essa con delle regole da rispettare, dove anche le virgole e gli spazi hanno un significato ben preciso. In questo paragrafo si propone una trattazione dettagliata della sintassi che governano le query SQL analizzate dall'algoritmo; questo aiuterà a capire l'idea su cui si basa questo progetto.

2.7.1 SELECT

La clausola SELECT è per forza di cose la prima ad essere individuata nella lettura di una query SQL. Come visto in precedenza, la sua presenza è obbligatoria ed è seguita dalla parola chiave FROM ,che ne individua le tabelle interessate.
La funzione di questa clausola è quella di elencare una serie di campi su cui si intende effettuare la ricerca, per poi visualizzarli nel risultato della query. Nell'elencare i campi di ricerca, è possibile rinominare il nome di uno o più elementi grazie al comando AS: il costrutto è evidenziato in figura 2.6.

In questo modo, all'interno della tabella risultato, il campo rinominato non presenterà più il suo nome originale ma quello seguente ad AS.
Infine si nota che l'elenco di tutti campi (con l'operatore AS o meno) viene effettuato uno di seguito all'altro, separandoli da una virgola: questo permette di

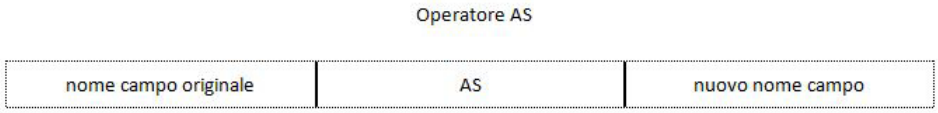

Figura 2.6: Costrutto dell'operatore AS

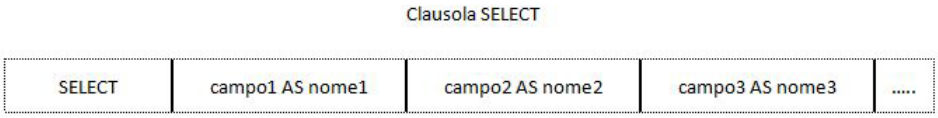

Figura 2.7: Suddivisione della clausola SELECT

definire con assoluta precisione dove finisce la dichiarazione di un campo e dove ne inizia un'altra.
La suddivisione della SELECT avviene quindi come mostrato in figura 2.7.

Va ricordato che non è necessaria la presenza dell'operatore AS: nel caso in cui questo manchi, il campo sarà individuato solo dal suo nome, che manterrà anche nella tabella risultato.
Vi possono essere degli operatori speciali all'interno di questa clausola; questi sono:

- DISTINCT: garantisce l'eliminazione dei record duplicati nel risultato della query;
- * (asterisco): indica che la SELECT prenderà tutti i campi contenuti nelle tabelle interessate.

2.7.2 FROM

La funzione della FROM è quella di elencare le tabelle coinvolte nella ricerca della query. Come per la clausola SELECT, anche qui è possibile trovare l'operatore AS per la ridenominazione delle tabelle coinvolte, anche se è pur sempre facoltativo.
Gli elementi della clausola sono qui collegati da un operatore che indica la relazione che esiste tra una tabella e quella successiva: l'operatore di JOIN. Tale

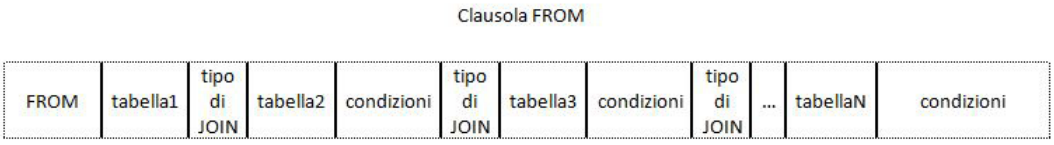

Figura 2.8: Sintassi base della clausola FROM

operatore serve a combinare le righe di due o più tabelle di un database.
Vi sono diversi tipologie di questo operatore, in relazione alla funzione di associazione che esso attua tra le tabelle coinvolte. Qui di seguito troviamo le principali:

- INNER JOIN: crea un nuova tabella combinando i valori delle due tabelle di partenza (A and B) basandosi su una certa regola di confronto (specificata come condizione). La query compara ogni riga della tabella A con ciascuna riga della tabella B cercando di soddisfare la regola di confronto definita. Quando la regola di join viene soddisfatta, i valori di tutte le colonne delle tabelle A e B vengono combinate in un'unica riga nella costruzione della tabella risultante;

- CROSS JOIN: effettua il prodotto cartesiano di tutte le righe delle tabelle che concorrono alla query di join;

- OUTER JOIN: funzione che non richiede che ci sia corrispondenza esatta tra le righe di due tabelle. La tabella risultante da una outer join trattiene tutti quei record che non hanno alcuna corrispondenza tra le tabelle. Le outer join si suddividono in left outer join, right outer join, e full outer join, in base a quale sia la tabella di cui intendiamo trattenere i valori in caso di mancata corrispondenza della regola di confronto (quella destra, quella sinistra o entrambe).

Spesso, seppur facoltativo, è possibile individuare anche delle condizioni che accompagnano gli operatori di JOIN: tali condizioni mi permettono di applicare delle regole per la combinazione dei record contenuti nelle tabelle al fine di ottenere i risultati richiesti dall'utente. Tali condizioni sono specificate nel seguente modo: viene indicata la seconda tabella coinvolta dall'operatore di JOIN, viene inserita la parola chiave ON e di seguito elencate tutte le condizioni da verificate nello svolgere l'operatore. La sintassi risulta essere espressa in figura 4.5.

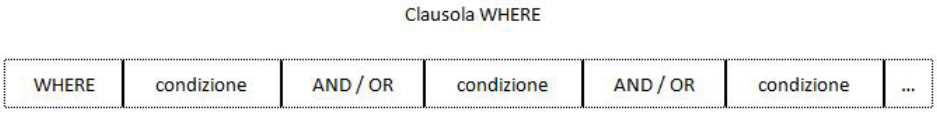

Figura 2.9: Sintassi base della clausola WHERE

Va sottolineato, come fatto nell'analisi della SELECT, che le caratteristiche elencate non sono tutte necessarie. L'uso dell'operatore di JOIN è d'obbligo, volendo effettuare delle operazioni sulle tabelle, ed alcuni di questi necessitano di specificare delle condizioni. Tuttavia è bene ricordare che non sempre sono necessarie, pertanto possono non essere specificate.
Un caso è quello di voler prendere i risultati analizzando i record di una sola tabella: in tal caso non vi sono né operatori di JOIN né condizioni.

2.7.3 WHERE

la clausola WHERE è una clausola facoltativa, ovvero può essere omessa dalla specifica della query ottenendo comunque un risultato valido. Poiché permette di selezionare i record da visualizzare a seconda che rispettino o meno certe condizioni, viene spesso utilizzata per ottenere un risultato più dettagliato.
La costruzione di tale clausola è molto semplice: si elencano le condizioni da voler applicare alla query specificandone le caratteristiche. Il costrutto sintattico delle condizioni è lo stesso per tutte, ovvero consiste in una semplice enumerazione delle condizioni, messe tra loro in relazione dagli operatori AND e OR. Come accennato in precedenza, l'ordine con cui vengono valutati è fondamentale per capire come sono messi in relazione tra loro: difatti se tra due condizioni sostituissimo un AND con un OR, il risultato finale della query risulterebbe compromesso.
La clausola WHERE con un elenco di condizioni si presenta tipicamente come indicato in figura 4.9, dove ciascuna condizione può essere costruita in vari modi: ci sono infatti sia operatori di confronto a valori singoli che a valori doppi.Le prime sono quelle che utilizzano un unico valore per effettuare confronti, gli altri invece utilizzano 2 valori per confrontare i dati delle colonne. In figura 2.10 si ha un esempio dei loro costrutti.

Gli operatori utilizzati possono essere di vario tipo: possono confrontare valori aritmetici, effettuate uguaglianze tra valori e molto altro ancora.

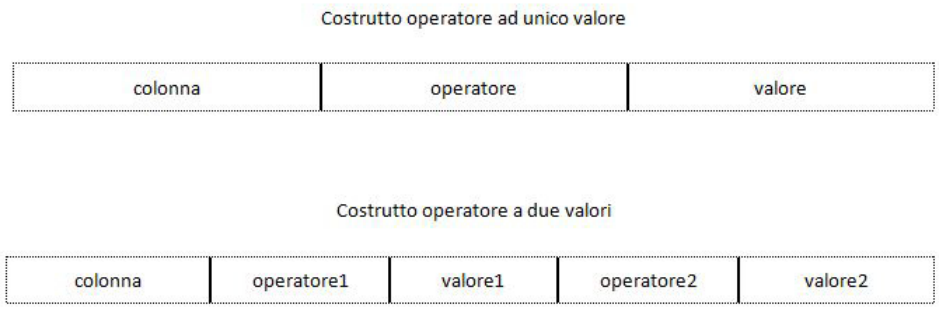

Figura 2.10: Costrutto di operatori a uno o due valori

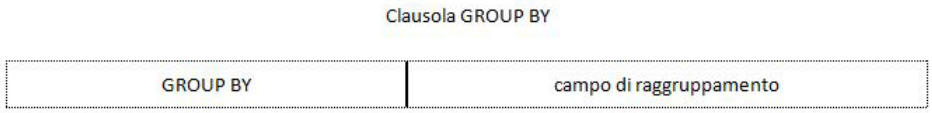

Figura 2.11: Clausola GROUP BY

2.7.4 GROUP BY

Si tratta di una clausola estremamente semplice da costruire: è necessario semplicemente specificare il nome del campo secondo cui si vuole effettuare il raggruppamento.

2.7.5 HAVING

La clausola HAVING, sebbene il suo impiego sia strettamente connesso alla clausola GROUP BY, è costruita nello stesso modo della clausola WHERE: presenta, dopo la parola chiave, un elenco di condizioni da verificare, collegate dagli operatori AND e OR.
Come sempre si fa notare che la clausola è facoltativa e che il numero di condizioni è arbitrario e va a seconda delle esigenze dell'utente.

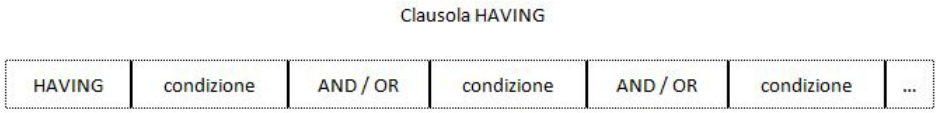

Figura 2.12: Clausola HAVING

Figura 2.13: Clausola ORDER BY

2.7.6 ORDER BY

Come per GROUP BY, basta indicare il campo secondo cui si vuole effettuare l'ordinamento, seguito inoltre dal tipo di ordinamento: se crescente utilizzo il codice ASC, se decrescente utilizzo il codice DESC. Per un esempio si veda figura 2.13.

Dalla relazione qui esposta si evidenzia subito un fatto importante, che verrà impiegato nell'analisi del progetto: le clausole presentano alcuni aspetti tra loro simmetrici, infatti i costrutti sono simili. La sintassi impone l'utilizzo di una parola chiave seguita da un elenco degli elementi specifici richiesti dal tipo di clausola, sino essi tabelle, campi o condizioni.
In particolare, le condizioni, in qualunque clausola esse siano riportate, si compongono sempre nello stesso modo, con un primo termine di confronto, l'operatore è il secondo termine di confronto. Ciò ha permesso una semplificazione nella loro scomposizione ed analisi, che si vedrà in dettaglio nel capitolo 5.

Capitolo 3

Le specifiche

In fase di sviluppo sono stati discussi ed individuati dei requisiti che il progetto deve rispettare per poter essere funzionale ed implementabile correttamente all'interno del prodotto EDP:

- Essere in grado di riconoscere tutto il patrimonio di query SQL formulabile dall' EDP;
- Valorizzare correttamente ogni elemento costitutivo del codice SQL;
- Interagire correttamente con gli input e gli output dell'EDP;
- Interfacciarsi coi metodi messi a disposizione dalla libreria DDPv2.

Per questo si è fatta molta attenzione al rispetto di tale requisiti, verificati man mano che si presentavano con opportuni test e con frequenti revisioni. Alcuni di questi requisiti sono stati anche estesi a possibili implementazioni future e migliorie alla suite aziendale:

- Analisi di altri tipi di stringhe di comando SQL;
- Analisi di query annidate;
- Analisi di query con operatori insiemistici;
- Estensione del dizionario degli operatori analizzabili.

Dovendo inserire l'algoritmo in una suite aziendale già collaudata, le specifiche richieste dall'azienda si basano soprattutto sulla possibilità di inserire il progetto all'interno del codice sorgente della loro suite aziendale. Fondamentali sono infatti la cattura del codice SQL della query in ingresso all'algoritmo e l'oggetto di tipo Query che verrà restituito in uscita all'EDP; quest'ultimo in particolare dovrà essere corredato di tutte le informazioni necessarie per poter ricostruire la query a livello grafico in modo assolutamente perfetto, altrimenti si avrebbe una sensibile perdita di informazioni nei vari passaggi tra ambiente grafico e testuale. Le query da analizzare inoltre sono state ben definite dall'azienda nei minimi particolari, precisando anche quali operatori si sarebbero potuti incontrare in fase di analisi al di fuori di quelli conosciuti dallo standard SQL-92. Inoltre, sono state ben evidenziate le capacità della suite aziendale nel recepire gli elementi costitutivi della query: le classi ed i metodi della suite aziendale che, a partire da valori semplici, riescono a ricostruire la query a livello grafico sono stati specificati e chiariti sin dall'inizio.
In merito a specifiche sulla complessità dell'algoritmo, sulle tecniche da utilizzare per lo sviluppo del progetto e sulla composizione del codice, è stata data completamente carta bianca. Pertanto le scelte effettuate sono state fatte in base alla semplicità delle soluzioni che si sono mano a mano presentate in fase di studio.

3.1 Tipi di istruzioni da analizzare

In figura 3.1 è riportata una tabella con tutte le specifiche riguardanti le istruzioni da analizzare.

3.2 Strumenti a disposizione

Di seguito vengono riportati gli strumenti messi a disposizione per la creazione dell'algoritmo. L'utilizzo del codice C#[1] è stato imposto dall'azienda, mentre l'utilizzo delle regular expression è solo stato consigliato. Il tipo di strutture dati per le memorizzazioni temporanee delle porzioni di stringa analizzate, l'idea base del progetto e la struttura delle classi e dei metodi sono stati scelti in corso d'opera su mia iniziativa.

[1] http://msdn.microsoft.com/it-it/library/kx37x362.aspx

Tipo	Descrizione
SQL	Il progetto si basa sul riconoscimento di stringhe SQL che rispettino il linguaggio standard SQL 92. Di conseguenza occorre utilizzare funzioni relative a tale linguaggio.
Sintassi	La sintassi per la scrittura o la modifica della stringa da analizzare deve essere precisa e corretta al fine di evitare errori nel programma o nell'analisi degli elementi.
Associazione dei valori	Ogni funzione del linguaggio è associata ad un relativo codice operativo, univoco per ogni operazione. In fase di creazione dell'oggetto, la relazione tra elemento e codice deve essere perfettamente rispettata.
Valutazione e riconoscimento dei principali elementi costitutivi della query	Le query devono essere necessariamente costituite dalle clausole SELECT e FROM per poter essere valide. In aggiunta, è possibile inserire condizioni e campi di raggruppamento o di ordinamento attraverso le altre clausole opzionali. La classe dovrà essere in grado di effettuare il riconoscimento e la scomposizione di tali clausole in maniera precisa e di saper riconoscere a quale tipologia esse appartengono.
Valutazione degli elementi	Ogni elemento costitutivo della stringa ha una esatta funzione, dovuta sia al ruolo all'interno della clausola sia in base al contesto in cui viene inserito. Occorre che il prodotto sappia riconoscere e valorizzare correttamente l'elemento trattato.
Rispetto del DDPv2	L'algoritmo del progetto come già detto risente fortemente dello sviluppo delle funzionalità dell'EDP. Per questo deve perfettamente integrare le sue funzionalità con quelle messe a disposizione della suite, e rispettare pertanto i vincoli che vengono posti agli oggetti in uscita e successivo ingresso al software principale.
Query di interrogazione	Il prodotto è stato sviluppato per poter interpretare solo ed esclusivamente query di tipo interrogativo. Pertanto non è in grado di analizzare query di update o modifica al database.
Molteplicità di inserimento	Le stringhe SQL possono essere inviate tramite input da console, file di testo o passate alla classe tramite variabili in formato string. Il progetto deve saper accettare in ingresso tali valori in qualunque modo essi si presentino.
Individuazione di query annidate	Poiché è frequente trovare query annidate, è necessario che il prodotto sappia riconoscerle e gestirle senza problemi. L'analisi approfondita dell'algoritmo però è indispensabile solo per la query di primo livello, non per quella annidata.
Eventuali operatori insiemistici	Come per le query annidate, è necessario che il programma sia in grado di riconoscere e valutare correttamente eventuali query collegate da operatori insiemistici, quali UNION, UNION ALL, EXCEPT e INTERSECT. Poiché però la suite aziendale non è in grado di gestire graficamente gli operatori insiemistici, non ne è richiesta un'implementazione aggiuntiva.

Figura 3.1: Specifiche delle istruzioni da analizzare

3.2.1 Linguaggio C#

Il C# è un linguaggio di programmazione object-oriented sviluppato da Microsoft[2] all'interno dell'iniziativa .NET[3], e successivamente approvato come standard ECMA[4]. La sintassi del C# prende spunto da quella del Delphi[5], del C++[6], da quella di Java[7] e da Visual Basic[8] per gli strumenti di programmazione visuale e per la sua semplicità : difatti presenta meno simbolismo rispetto a C++ e meno elementi decorativi rispetto a Java. C# è, in un certo senso, il linguaggio che meglio degli altri descrive le linee guida sulle quali ogni programma .NET gira; questo linguaggio infatti è stato creato da Microsoft specificatamente per la programmazione nel Framework .NET. I suoi tipi di dati primitivi hanno una corrispondenza univoca con i tipi .NET e molte delle sue astrazioni, come classi, interfacce, delegati ed eccezioni, sono particolarmente adatte a gestire il .NET framework. Come Java ha i suoi package, anche nel C# possiamo ritrovare una serie di classi già sviluppate per l'interazione con i vari ambienti, Front End, Database, Xml e altri. Questo è il .NET framework, del quale utilizza una serie di librerie di classi che gli permettono l'accesso alle funzionalità del sistema. In C# quello che in Java è chiamato package viene chiamato namespace o spazi di nomi. Le classi sono organizzate all'interno di una serie di namespace che raggruppano le classi con funzionalità simili; ad esempio System.Windows.Forms per la gestione delle finestre di tipo Forms, System.Xml per l'elaborazione di XML e System.Data per l'accesso alle basi dati.

3.2.2 Strutture dati

Sono state utilizzate due strutture dati in particolare:

- QUEUE: spesso è stato necessario memorizzare temporaneamente dati in strutture che memorizzassero anche l'ordine di inserimento; difatti la composizione delle stringhe FROM e WHERE, ad esempio, sono caratterizzate dall'associazione tra loro di più tabelle o di più condizioni attraverso speciali

[2]http://www.microsoft.com/
[3]http://www.microsoft.com/net/
[4]http://www.ecma-international.org/
[5]http://www.embarcadero.com/products/delphi/
[6]http://www.cplusplus.com/
[7]http://www.java.com/
[8]http://msdn.microsoft.com/it-it/vbasic/default.aspx

operatori (in questo caso JOIN, AND, OR e altri) che valutano, secondo un ordine ben preciso, tutti gli elementi costitutivi del codice. Per questo, nell'estrazione di tali elementi, bisognava poi cercare di ricostruirli con lo stesso esatto ordine con cui erano stati separati e valorizzati. Ciò spiega il ricorso alle code (o queue): dato che la politica di inserimento e rimozione è di tipo FIFO (ovvero quello che viene inserito per primo è il primo ad essere estratto), si ha avuto la possibilità di utilizzare una struttura dati che tenesse conto dell'ordine di inserimento e di estrazione corretto per gli elementi analizzati. La queue nel linguaggio C# è definita come una struttura dati di tipo astratto implementata grazie ad un array opportunamente ridimensionabile;

- ARRAY: per la memorizzazione rapida di pochi elementi dove ognuno di questi dovesse essere valorizzato in modo diverso, si è scelto l'impiego degli array come struttura dati. Spesso per la memorizzazione delle colonne e dei relativi alias nella clausola SELECT, si è visto che le code potevano rappresentare un rischio nell'analisi dei singoli elementi: difatti gli elementi all'interno di una coda vengono si memorizzati tenendo conto dell'ordine, ma un inserimento o un estrazione errati potrebbero compromettere l'analisi del codice. L'array invece presenta dimensione fissa, e in ognuna delle sue celle possono essere inseriti elementi valorizzati in modo diverso in seguito senza errori e più facilmente riconoscibili all'interno del codice.

3.2.3 Regular Expression

Le Espressioni Regolari sono sintassi attraverso le quali si possono rappresentare insiemi di stringhe. Gli insiemi caratterizzabili con espressioni regolari sono anche detti linguaggi regolari (e coincidono con quelli generabili dalle grammatiche regolari e riconoscibili dagli automi a stati finiti).
Più rigorosamente possiamo definire un'espressione regolare, a partire da un alfabeto A ed un insieme di simboli Z come la stringa R appartenente a (A U Z) tale che valga una delle seguenti operazioni:

- R = insieme vuoto;
- R appartiene ad A;

- R = S+T oppure R=ST oppure R=S*, dove S e T sono espressioni regolari sull'alfabeto A.

Le espressioni regolari sono utilizzate principalmente da editor di testo per la ricerca e la sostituzione di porzioni del testo. Tuttavia i linguaggi come le espressioni regolari sono adatti a rappresentare un ristrettissimo insieme di linguaggi formali: l'utilizzo dei linguaggi regolari è comunque conveniente, in quanto la chiusura degli stessi alle operazioni di unione, intersezione e complementazione, permettono la costruzione di un'algebra di Boole ed una buona decidibilità.
Sono state utilizzate all'interno di questo progetto al fine di rendere più facile la ricerca delle parole chiavi all'interno della stringa SQL; inoltre, grazie alla possibilità di definire con esattezza anche più di una parola, è stato facile trovare anche parole chiave composte.
In alcuni punti del codice, tuttavia, si è dovuti procedere al comune algoritmo di pattern matching, a causa della difficoltà delle regular expression nella ricerca di singoli caratteri di punteggiatura.

3.3 Interfacciamento con plain Extended Data Publisher

Uno dei requisiti fondamentali che QueryAnalyzer e LibQueAnalyzer (d'ora in poi abbreviate rispettivamente con QA e LQA) devono avere è la perfetta integrazione con l'EDP: le specifiche richiedono, in entrata al progetto, una stringa di codice SQL rappresentante la query da analizzare che può essere inserita manualmente via console oppure come variabile di tipo String dalla suite aziendale. Per quanto riguarda il prodotto di uscita, l'algoritmo deve essere in grado di costruire un oggetto contenente tutte le caratteristiche e proprietà rilevate dalla scomposizione della query stessa: tale oggetto è definito dalla classe DDPv2 e viene definito come oggetto di tipo Query. Per una verifica dell'analisi tale oggetto verrà poi specificato in un file XML tramite cui sarà possibile leggerne le caratteristiche costruttive.

3.3.1 Ingresso

Nel caso si utilizzi LQA, trattandosi di una libreria e non di un applicativo a se stante, il problema non sussiste, in quanto sarà il programma principale a

chiamare la funzione necessaria per leggere la query. Nel caso di QA invece, il programma effettua una richiesta all'utente mirata a capire se l'intenzione sia quella di inserire manualmente la query oppure se si voglia inserirne l'indirizzo di memoria in cui andare a leggerla.
Per effettuare la scelta l'utente dovrà scegliere tra due codici specifici, indicati nella prima riga di commando, e indicarli come input. Qualora scegliesse l'inserimento manuale, dovrà poi digitare manualmente il codice della query in formato SQL da voler analizzare, mentre scegliendo la ricerca da indirizzo di memoria l'utente sarà poi obbligato all'inserimento del percorso in cui si trova il file contenente la query in questione.

3.3.2 Uscita

Per entrambe le versioni del progetto, l'uscita è caratterizzata da un oggetto di tipo Query.Tale oggetto viene composto gradualmente in fase di analisi: deve infatti contenere riferimenti a tabelle utilizzate, campi da evidenziare, codici di identificazione per operatori matematici e logici e condizioni da verificare. Tutte queste specifiche vengono registrate nell'oggetto attraverso l'utilizzo di metodi della classe DDPv2. Per ogni clausola vengono esaminati tutti gli elementi e, uno alla volta, valorizzati nel modo corretto e registrati nell'oggetto di tipo Query: cosi si avrà un'aggiunta delle caratteristiche componenti la query alla fine dell'analisi di ogni sua parte.
Ecco i metodi principali (che si vedranno più dettagliatamente nel codice) che servono alla costruzione dell'oggetto Query indicato dalla variabile q:

- Inserimento campo: q.AddField(counter, codeName, codeName, formula, order, funcode, 0, true, false, false, string.Empty, 0, 0);
- Inserimento tabella: q.Tables.Add (new Tabella(asName, codeName, 0, 0, 0, 0, string.Empty, -1, 0)) la tabella viene creata e poi inserita come oggetto valorizzato nella Query q;
- Inserimento condizioni: q.AggiungiCondizioni(q.User, arCond) dove arCond è un array contenente tutte le condizioni espresse come oggetti di tipo Condizione.

Si ricordi che ogni parametro inserito nei metodi sopra citati viene valutato dall'algoritmo ed opportunamente inserito.

Dopo la terminazione dell'analisi, l'oggetto viene finalizzato e restituito come parametro d'uscita al processo chiamante.

Capitolo 4

Implementazione dell'algoritmo

In questo capitolo viene esemplificato l'algoritmo adottato per lo sviluppo del progetto. I fondamenti del linguaggio SQL chiariti in precedenza saranno utili per meglio comprendere le scelte effettuate per l'analisi della query e delle clausole in essa contenute. Verrà inoltre evidenziato come sono stati rispettati tutti i requisiti richiesti dall'azienda e come l'algoritmo si comporta in relazione alle query complesse, quali quelle annidate o quelle con operatori insiemistici.

Il processo che sta alla base del riconoscimento e scomposizione della stringa SQL si basa sull'osservazione del costrutto sintattico che la stringa presenta. Poiché la sintassi utilizzata è standard e ben definita, le regole applicate per l'analisi saranno valide per tutte le query soggette ad analisi. Nel caso in cui una stringa venga digitata in modo errato non rispettando la sintassi SQL, verrà riconosciuta come errata e pertanto non analizzata.
Tenendo conto di tutte le regole viste, si è scelto di operare una analisi della stringa SQL secondo diversi livelli in cui, ad ogni passo, la query viene scomposta in elementi sempre più semplici e diversificati. Ogni elemento individuato, a seconda della sua natura, verrà riconosciuto e processato in modo mirato. Ottenute le parti elementari di cui è composta la stringa SQL, si procede alla costruzione dell'oggetto di tipo Query: per fare questo ci serviremo della classe DDPv2 che contiene tutte le specifiche per la costruzione dell'oggetto necessario alla creazione della query a livello grafico.

4.1 Individuazione della query

Come prima cosa l'algoritmo si occupa di ricevere in ingresso la stringa SQL da analizzare; essa può essere inserita a mano dall'utente oppure individuata da un indirizzo di memoria in cui risiede. Nel primo caso la stringa viene presa direttamente dall'ingresso di input standard (o via console) e memorizzata all'interno di un parametro di tipo String. Nel secondo caso viene preso l'indirizzo di memoria, viene individuato e letto il file che la contiene e viene memorizzata nel parametro di tipo String.
L'applicazione di queste due metodologie avviene in modo diverso a seconda dell'applicativo scelto:

- QA permette l'inserimento manuale o del codice SQL rappresentante la query o di un indirizzo di memoria in cui andare a ricercarla;
- LQA riceve automaticamente la variabile di tipo String contenente il codice SQL della query dall'EDP.

4.2 Analisi di query annidate e operatori insiemistici

Il primo livello di analisi consiste nel verificare il costrutto della query: nel caso siano presenti una o più query annidate, sarà necessario rimuoverle e processarle separatamente sostituendovi il relativo codice con una particolare parola chiave; allo stesso modo vengono valutati eventuali operatori insiemistici e le query ad essi collegate.

La prima analisi effettuata è quella relative alle query annidate:

- Il programma principale passa ad un metodo della classe del progetto la variabile di tipo String in cui la query è stata inserita e il riferimento di una coda contenente dati di tipo String che memorizzerà temporaneamente eventuali query annidate;

- Dalla sintassi si evince che le query annidate sono racchiuse tra "(" e ")" e caratterizzate dalla formula costruttiva composta almeno da SELECT e FROM: quindi l'inizio di una eventuale query annidata è dato da una costruzione definita da "(SELECT" o eventualmente "(SELECT". Il secondo costrutto è stato realizzato in modo da poter trattare anche eventuali spazi tra il simbolo e l'inizio della parola chiave. Costruendo una espressioni regolari composta da tali forme chiave, sarà possibili ricerca all'interno della stringa SQL la presenza di una eventuale sottoquery e, in caso di esito positivo, estrarla ed inserirla all'interno della coda che le memorizza;

- Dato che la query annidata inizia con il simbolo "(" sarà necessario individuare il relativo simbolo ")" che ne indica la terminazione; una volta individuata, grazie ai metodi della classe String, si può procedere alla manipolazione della stringa;

- Viene rimosso il codice relativo alla query annidata e al suo posto viene inserita la parola chiave "###subquery###", che ne indicherà la presenza nell'analisi della clausola in cui è inserita e il punto in cui reinserirne il codice in fase di ricostruzione.

Il metodo è ricorsivo: poiché infatti le query annidate possono essere molteplici, il metodo continua l'analisi sopra descritta sino alla fine della stringa. Al termine restituirà al programma principale la stringa SQL priva di query annidate e la coda di tipo String che contiene le stringhe relative alle query annidate.
Nel caso in cui non vi siano presenti query di questo tipo, la stringa SQL non verrà modificata e la coda sarà vuota, non influenzando in alcun modo il corretto funzionamento dell'algoritmo.

Successivamente viene effettuata la ricerca di eventuali operatori insiemistici, sempre ricorrendo alle espressioni regolari: nel caso in cui la ricerca dia esito positivo, si spezza la query generale in due query memorizzate entrambe in due variabili di tipo String, una sarà quella che precede l'operatore insiemistico e una sarà quella che lo segue. L'analisi richiesta deve essere fatta solo sulla query che precede l'eventuale operatore insiemistico, pertanto verrà usata solo la variabile relativa alla prima query.
La classe principale, nel caso trovasse uno degli operatori, stamperà a schermo il tipo di operatore individuato segnalandone all'utente la presenza.

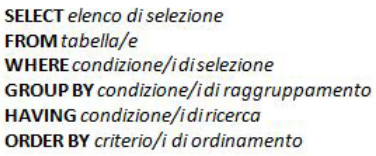

Figura 4.1: Costruzione generia di una query

Effettuata questa prima analisi, si procede al primo livello di scomposizione della stringa.

4.3 Scomposizione nelle clausole principali

Dopo le prime analisi e le eventuali modifiche si è ora in possesso di una query in codice SQL priva di query annidate e collegamenti con altre query tramite gli operatori insiemistici: è quindi pronta ad un primo livello di analisi.
Dai formalismi SQL, si sa che una query per essere valida deve essere costituita dalle clausole SELECT e FROM ed eventualmente dalle clausole accessorie WHERE, GROUP BY, ORDER BY e HAVING. Per questo ci si deve aspettare la presenza di tutte queste parole chiave e dei loro attributi.
Oltre alla loro presenza, devono essere anche inserite secondo un certo ordine; il tutto è evidenziato in figura 4.1.

La prima divisione si può operare andando a ricercare all'interno della stringa tali parole chiave, in modo da poter individuare in modo univoco le 6 clausole principali.
Individuando all'interno della stringa le parole chiave con cui le varie clausole iniziano, è possibile tracciare i limiti di ogni singola clausola, potendole così suddividere correttamente; tramite le espressioni regolari, si cercano le parole chiavi all'interno della stringa, si valutano gli indici in cui compaiono e successivamente si spezza la stringa in altre 6 stringhe inserite, sempre in ordine, all'interno di un array di tipo String in cui ogni cella è riservata ad una particolare clausola.
In questo modo, quando si andrà ad analizzare ciascuna delle stringhe appena divise, si sa che in base alla cella in cui si andrà a prenderle si dovrà procedere ad una analisi diversa e mirata al tipo di stringa trattata.
Nel caso in cui l'istruzione contenga tutte le clausole sopra citate l'array sarà

Tipo	Elementi	Sintassi
SELECT	Campi con eventuali alias o possibili funzioni	Enumerazione separata da virgole
FROM	Tabelle con eventuali alias unite tramite operatori di JOIN di vario tipo con possibili condizioni	Enumerazione delle tabelle collegate da JOIN
WHERE	Condizioni sui campi	Enumerazione dove ogni condizione è collegata da "AND" o "OR"
ORDER BY	Campi o alias per gli ordinamenti	Enumerazione separata da virgola
GROUP BY	Campi o alias per i raggruppamenti	Enumerazione separata da virgola
HAVING	Condizioni di ricerca	Enumerazione dove ogni condizione è collegata da "AND" o "OR"

Figura 4.2: Elementi e sintassi delle varie clausole

pieno, altrimenti l'assenza di qualche elemento lascerà le relative celle vuote: in tal caso si conferma l'assenza di tali clausole e si procede con l'analisi degli attributi.

4.4 Analisi di ogni clausola

Ottenuti gli elementi costitutivi generici di una query, si può procedere all'analisi delle loro parti costitutive: dato l'array contenente le clausole, una ad una, vengono estratte ed analizzate con un metodo loro riservato, effettuando quindi un secondo livello di analisi.
Dal punto di vista sintattico, si ricorda che le clausole sono costruite secondo lo schema di figura 4.2.

Data la somiglianza della sintassi di SELECT, ORDER BY e GROUP BY, queste clausole sono state analizzate assieme; inoltre per la costruzione dell'oggetto Query l'inserimento dei campi SELECT coincide, in fase di inserimento, con quello degli attributi di ordinamento e raggruppamento, e per questo va considerata come una unica operazione.
All'interno di QA e di LQA sono stati creati dei metodi che, tenendo conto della tipologia e del costrutto delle varie clausole, riescono ad analizzare ed individuare correttamente tutti gli elementi costitutivi del codice.

I metodi citati in figura 4.3 hanno un primo funzionamento in comune: dall'analisi delle query in SQL prodotte dalla suite aziendale EDP, spesso tabelle

SELECT, ORDER BY e GROUP BY	analizzate da	*analyzeSelFields*
FROM	analizzata da	*analyzeFromTable*
WHERE	analizzata da	*analyzeWhereCondition*
HAVING	analizzata da	*analyzeHavingCondition*

Figura 4.3: Metodi relativi all'analisi

SELECT	elementi così composti	*campo* AS *alias*
ORDER BY	elementi così composti	*campo* ASC/DESC
GROUP BY	elementi così composti	*campo*

Figura 4.4: Composizione elementi SELECT, ORDER BY e GROUP BY

e campi venivano racchiusi tra apici; in fase di ricostruzione, il DDPv2 non era in grado di riconoscere all'interno del database le tabelle o i campi limitati dagli apici. Per questo è stato inserito un metodo che, prima di ogni analisi, sia in grado di eliminare gli apici dal codice: data la stringa, usando l'algoritmo di pattern matching tutta la punteggiatura inutile viene sostituita con un carattere vuoto, portando il codice ad una forma analizzabile. Una volta effettuata questa pulizia, i vari metodi procedono all'analisi degli elementi.

4.4.1 analyzeSelFields

Il metodo prende le clausole SELECT, ORDER BY e GROUP BY e ne effettua la scomposizione in termini semplici: gli attributi di tali clausole sono separati tra loro dalle virgole, pertanto questa peculiarità verrà sfruttata per dividere i campi tra loro.
Dividendo queste stringhe a seconda della posizione delle virgole, otteniamo un insieme di elementi costruiti secondo lo schema di figura 4.4.

Tali elementi verranno inseriti prima in una coda temporanea e in seguito, una volta analizzati, trasferiti su un array per poterli valorizzare uno ad uno. Si nota che, a seconda della clausola in cui sono inseriti, sebbene siano tutti dei campi possono essere seguiti dall'operatore AS e il nuovo nome oppure dalla dicitura ASC o DESC. Prelevando la stringa relativa a SELECT, per ogni elemento verrà individuato in ordine il nome originale del campo, l'eventuale operatore AS e l'eventuale nuovo nome del campo posizionato dopo AS; verranno poi inseriti in un array di 3 elementi in cui ogni cella ne contiene una specifica porzione.
Anche le clausole ORDER BY e GROUP BY vengono sottoposte allo stesso trattamento, con la differenza che in questo caso la prima potrebbe presentare dei

codici di ordinamento, mentre la seconda solo il nome del campo. Il significato di questi campi è tuttavia diverso dagli altri: questi indicano una funzione di ordinamento o raggruppamento da applicare ai campi individuati nella SELECT. L'algoritmo, man mano che analizza i campi della clausola SELECT, provvede a ricercare contemporaneamente la loro presenza all'interno di ORDER BY E GROUP BY e nel caso li individui, grazie ai metodi della classe DDPv2, aggiunge tali funzionalità ai relativi campi mediante particolari codici identificativi sempre definiti nella classe DDPv2 che vanno rispettati:

- se la ricerca di un elemento nella clausola ORDER BY è positiva, l'algoritmo provvederà a costruire l'elemento nell'oggetto Query indicando nella variabile relativa all'ordinamento il codice ASC nel caso l'ordinamento sia crescente, DESC nel caso sia decrescente;
- se la ricerca di un elemento nella clausola GROUP BY è positiva, l'algoritmo provvederà a costruire l'elemento nell'oggetto Query indicando nella variabile relativa al raggruppamento il valore 2 che ne indicherà la funzione di raggruppamento;
- nel caso siano verificate entrambe, verranno modificate entrambe le variabili, se al contrario non fosse verificata neanche una delle due, le relative variabili resterebbero inizializzate a valori di default indicando l'assenza di tali funzioni.

Sempre all'interno della clausola SELECT, è possibile trovare delle espressioni contenenti funzioni matematiche, ad esempio effettuanti somme, medie, minimi, massimi e altro ancora. Grazie alle espressioni regolari, viene effettuata la ricerca di tali operatori e, in caso di successo, il campo in cui la formula è presente viene inserito all'interno dell'oggetto come fosse una funzione e non un campo.
Al termine della valutazione del singolo elemento, l'algoritmo ne inserisce i campi all'interno dell'oggetto di tipo Query ricorrendo al metodo AddFields, sempre della classe DDPv2, in cui verranno specificati il nome originale del campo, il suo eventuale alias, l'eventuale codice di ordinamento e l'eventuale codice identificativo per il campo di raggruppamento.
Iterando tale procedimento per tutte le porzioni di codice individuate sino alla fine dell'array, completando l'analisi della clausola SELECT.

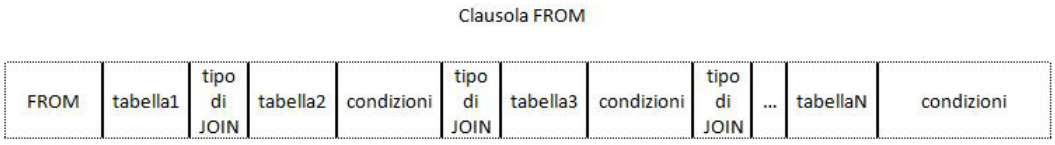

Figura 4.5: Sintassi base della clausola FROM

4.4.2 analyzeFromTable

Il metodo si occupa dell'analisi e successivo inserimento degli elementi della clausola FROM, ovvero le tabelle e le relazioni che intercorrono tra loro. A differenza degli elementi della clausola SELECT presentano differenze tipologiche sostanziali ma sintatticamente una struttura molto simile. Una clausola FROM può contenere da una a molte tabelle che vengono tra loro messe in relazioni dall'operatore di legame JOIN: si nota che, separando gli elementi costitutivi della stringa in oggetto a seconda della posizione dei vari JOIN, si riescono ad ottenere delle unità sintattiche rappresentanti ciascuna tabella coinvolta e le sue eventuali condizioni di join. Successivamente, la scomposizione delle unità in frammenti più piccoli ci permette di trovarne gli elementi costitutivi più semplici e di valorizzarli uno ad uno.
Una prima operazione da effettuare è quella di depurare il codice da eventuali segni di punteggiatura, e viene utilizzata la stessa procedura vista per analyzeSelFields.
Viene poi effettuata un'ulteriore sostituzione: spesso le query processate dall' EDP sostituiscono, a livello di codice SQL, l'operatore CROSS JOIN con la virgola. Poiché la sintassi SQL per la clausola FROM non prevede l'uso di virgole,il metodo effettua quindi la sostituzione di tutte le virgole presenti della stringa con l'operatore CROSS JOIN.
Ricordando la teoria del codice SQL si evince che la stringa della clausola FROM sarà realizzata seguendo il costrutto di figura 4.5, ricordando che le condizioni sono elementi opzionali.

La prima tabella della clausola non può contenere condizioni di join, in quanto tali condizioni vengono inseriti al termine della seconda tabella della relazione se presenti.
Fatta esclusione quindi per la prima tabella, gli altri elementi, dividendo la stringa secondo la posizione dei vari JOIN, presenteranno la forma mostrata in figura 4.6.

Figura 4.6: Elementi scomposti secondo il JOIN

Figura 4.7: Singoli elementi della clausola FROM

Il tipo di JOIN, come visto nella trattazione teorica, può essere composto da una, due o anche tre parole chiave. Sempre grazie all'utilizzo delle espressioni regolari, ogni tipo di JOIN è di facile individuazione e rappresenta il limite sintattico della stringa per suddividerla in elementi più semplici.
Come visto, il primo elemento sarà privo delle condizioni di JOIN e verrà pertanto analizzato singolarmente. Tutti gli altri, presentando la stessa costruzione, vengono analizzate nello stesso modo con un procedimento iterativo. Inizialmente, data la stringa FROM completa e priva di punteggiatura superflua, si ricercano le parole chiave JOIN all'interno del testo: poiché sono considerati i separatori principali degli elementi costitutivi della clausola, la stringa viene spezzata prima della parola chiave e subito dopo di essa. Ripetendo il procedimento per tutti i JOIN, si ottengono componenti del codice caratterizzati dalla tabella, dalle eventuali condizioni e dalle prime parole costituenti l'operatore di JOIN ma non da tale parola chiave. Questi elementi verranno inizialmente inseriti all'interno di una coda di tipo String, per praticità, per poi essere trasferiti uno ad uno in una cella di un array.
Il secondo livello di analisi consiste nel ridurre ulteriormente la complessità degli attributi presenti all'interno della coda; il loro costrutto si presenta tale a quello della figura 4.7.

Si ricerca il tipo di JOIN ricorrendo alle espressioni regolari, se ne verifica il tipo e, in relazione ad esso, si memorizza all'interno di una variabile un carattere alfanumerico indicante all'oggetto Query il tipo di legame che lega le tabelle; infine si rimuove ogni residuo dell'operatore individuato. La stringa ora si presenta prima del tipo di JOIN.
L'obiettivo è ora quello di valorizzare ciascuno degli elementi più semplici del codice.Per rendere ciò possibile, si cercano all'interno del codice gli operatori AS

	Tipo	Variabile
Elemento antecedente AS	Nome della tabella	codeName
Elemento tra AS e ON	Eventuale nome dell'alias relativo alla tabella	asName
Elemento successivo ad ON	Eventuali condizioni di JOIN	onField

Figura 4.8: Variabili tabella, alias e condizioni

e ON, indicanti l'eventuale alias della tabella e le condizioni applicate sull'operatore di JOIN. Facendo sempre ricorso alle espressioni regolari, ove presenti, è possibile ottenere l'indice della stringa in cui compare sia l'operatore AS che l'operatore ON: a questo punto si ottengono i tre campi che verranno memorizzati in tre variabili dedicate evidenziate nella tabella 4.8.

Si ricorda che la presenza dell' alias o della condizione non è obbligatoria: nel caso in cui manchino entrambe o una delle due, le relative variabili saranno vuote, e in fase di composizione dell'oggetto Query non creeranno problemi.
Un'ultima analisi spetta alle eventuali condizioni di JOIN: precedute dalla parola chiave ON, si presentano come elenco di condizioni collegate tra loro dagli operatori AND o OR. Si effettua quindi una suddivisione della condizione a seconda di questi termini e tramite il metodo della classe DDPv2 vengono costruite le condizioni di JOIN per essere successivamente inserite all'interno dell'oggetto di tipo Query.
Una volta completata l'analisi ed inizializzate le variabili ai valori individuati, vengono chiamati i seguenti metodi appartenenti alla classe DDPv2:

- q.AddTabella(asName, codeName): prepara l'inserimento di una tabella all'oggetto Query di variabile q;
- Tabella tab = new Tabella(asName, codeName, 0, 0, 0, 0, string.Empty, -1, 0): inizializza la tabella creata ai valori definiti dall'analisi e ad un insieme di valori standard;
- q.Tables.Add(tab): aggiunge la tabella creata all'oggetto Query q;
- Relazioni joinRel = new Relazioni(): crea un oggetto di tipo relazioni definito nel DDPv2 per la realizzazione dell'operatore di JOIN;

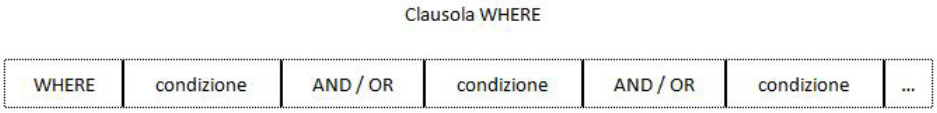

Figura 4.9: Sintassi base della clausola WHERE

- joinRel = buildRelation(asNamePrev, asName, code[counter - 1].ToString()): costruisce la relazione di JOIN a seconda dei valori individuate nell'analisi;
- q.Relazioni.Add(joinRel): aggiunge il tipo di relazioni all'oggetto di tipo Query q.

4.4.3 analyzeWhereCondition

E' il metodo che si occupa dell'analisi delle condizioni elencate all'interno della clausola WHERE. Come per le eventuali condizioni della tabella FROM, il principio che si applica per la scomposizione in elementi semplici degli elementi è sempre basato sugli operatori AND e OR: sono i separatori principali degli elementi di questa clausola.
La sintassi di tale clausola è mostrata in figura 4.9.

Vi è un aspetto molto importante di cui tenere conto: a differenza della semplice enumerazione vista nella clausola SELECT, nel caso delle condizioni è importante l'ordine con cui si presentano. Per questo motivo si è scelto di usare la coda come struttura dati per la memorizzazione degli elementi: effettuando una ricerca all'interno del codice delle parole chiave AND e OR, si ottengono i limiti sintattici di costruzione delle singole condizioni. Vengono inserite nella coda, in ordine, la condizione singola appena scomposta e l'operatore logico che la segue; iterando il procedimento, si riuscirà ad avere una coda in cui sono memorizzate alternativamente condizione e operatore.
Nel linguaggio SQL sono presenti anche un tipo particolare di operatori che utilizzano due parole chiave: uno di questi è il costrutto BETWEEN - AND. Per poterlo riconoscere è necessario effettuare l'analisi del codice prima di ogni altra operazione ed in modo leggermente diverso: prima si effettua la ricerca dell'operatore BETWEEN; nel caso di esito positivo, si deve procedere alla ricerca dell'operatore AND complementare al BETWEEN. E' di fondamentale importanza trovare l'AND esatto, altrimenti si rischia di compromettere il codice.

Per questo nel codice è stata inserita una variabile di tipo Boolean chiamata btw, inizializzata al valore false che indica la presenza o meno di tale operatore.
Nel caso in cui venga riscontrato un BETWEEN, la variabile viene posta al valore true.
La ricerca della parole chiave AND viene effettuata nello stesso modo, solo che oltre alla presenza di tale operatore viene valutato anche il valore della variabile btw: nel caso in cui sia posta al valore true, significa che l'AND appena trovato è quello relativo al suo complementare BETWEEN: in tal caso, non si scompone la stringa in quanto questo è un particolare tipo di condizione e non un operatore logico.

E' possibile all'interno del codice individuare un insieme di condizioni racchiuso dai simboli "(" e ")"; in questo caso, l'algoritmo dovrà tener conto della presenza di un eventuale simbolo "(" o ")" all'interno della singola condizione per poterla segnalare correttamente al DDPv2.
Scomposti gli elementi della clausola in parti più semplici, basterà valutare la singola condizione: viene controllata la presenza di un eventuale simbolo e ne viene segnalata la presenza in una apposita variabile a seconda che sia "(" o ")".
In seguito, grazie alle espressioni regolari, si cercano all'interno del codice gli operatori delle condizioni: a questo punto, eventuali operatori a doppia parola chiave non presentano più gli stessi problemi di prima, in quanto gli unici And che troveremo saranno quelli complementari a BETWEEN o NOT BETWEEN.
Con l'aiuto delle espressioni regolari, si individuano gli indici in cui compaiono gli operatori, si opera la suddivisione del codice nei suoi elementi costitutivi prendendone la parte prima dell'operatore e la parte successiva. In base agli operatori individuati, il metodo associa ad ogni operatore un codice numerico utile per la futura implementazione dell'oggetto Query.
Scomposta la condizione, i suoi elementi costitutivi vengono inseriti all'interno di una coda nel seguente ordine: campo, codice operatore, primo valore e secondo valore; quest'ultimo è presente solo nel caso si utilizzino operatori a doppia parola chiave (e di conseguenza a doppio valore).

Il metodo, per effettuare la costruzione della clausola WHERE all'interno dell'oggetto Query, invoca i seguenti metodi:

- List(Condizione) elencoCondizioni = new List(Condizione)(): serve a creare la lista di condizioni per la ricostruzione della query;

- Condizione c = new Condizione(): serve alla creazione di una nuova condizione. L'oggetto è di tipo Condizione definito dal DDPv2: per completare tale oggetto è necessario specificare gli attributi da inserire nella condizione tramite il costrutto variabile.attributo e inizializzandoli al valore ricavato dall'analisi della condizione, quali campi, codici operatori, primo e secondo valore, presenza di simboli e altri valori standard;

- elencoCondizioni.Add(cond): si aggiunge la condizione creata ad un elenco Condizioni, tipo di dati del DDPv2 che memorizza le condizioni della query;

- Condizione[] arCond = elencoCondizioni.ToArray(): trasforma l'elenco delle condizioni in un array;

- q.AggiungiCondizioni(q.User, arCond): aggiunge le condizioni all'oggetto Query specificando l'array che le contiene ed il codice utente;

- q.NormalizeWhere(): rende la clausola pronta per la ricreazione grafica.

4.4.4 analyzeHavingCondition

Il metodo opera in modo identico ad analyzeWhereCondition: Poiché questi metodi non erano dipendenti l'uno dall'altro, per evidenziare meglio il codice si è deciso di implementarlo a parte, richiamando spesso le funzioni di analyzeWhereCondition.

Poiché i metodi accessori del DDPv2 per la costruzione delle clausole costitutive della query sono già stati eseguiti, al termine dell'analisi delle clausole il nostro oggetto è pronto ad essere inizializzato: grazie al metodo q.SetQueryFormale(new UserApplicativo(nomeutente, 1, A, 1)) dove q è l'oggetto di tipo Query creato.
Si è inserita la possibilità di produrre un file xml con le specifiche della query ricreata in modo da poterne valutare la correttezza.

4.5 Architettura

Mentre per LQA tutto è stato inserito in una unica classe principale, QA è costituito da 4 classi:

- MainClass, che include il metodo Main principale ed il metodo per l'analisi di una eventuale query annidata;
- Analyze, che racchiude i principali metodi che provvedono alla scomposizione della stringa nelle clausole principali della sintassi SQL, ovvero SELECT, FROM, WHERE, GROUP BY, HAVING e ORDER BY, all'individuazione di una eventuale query annidata, e all'individuazione di eventuali operatori insiemistici tra query;
- Clause, attraverso quattro metodi principali e svariati metodi accessori, analizza i vari campi della stringa ad hoc secondo il tipo di elementi e ricostruisce l'oggetto di tipo Query che verrà poi elaborato dal DDPv2 e riconvertito in forma grafica;
- File Manager, che permette la creazione dei file XML necessari per verificare la corretta creazione dell'oggetto Query.

Tralasciando la classe FileManager, utile solo alla creazione dello script SQL, si è scelta questa architettura per emulare visivamente il comportamento del progetto realizzato: infatti la classe Analyze procede ad una prima scomposizione, ad alto livello, della stringa data mentre la classe Clause provvede ad elaborare in dettaglio ogni minimo elemento della stringa, ed a valorizzarlo in modo corretto. Nelle figure 4.10, 4.12 e 4.13 verranno brevemente illustrati in dettaglio i metodi più importanti utilizzati per la costruzione del prodotto.

Classe MainClass

Oltre al metodo Main, è presente il metodo nestedQueryAnalyzer che, comportandosi allo stesso modo del metodo Main, analizza le eventuali query annidate.

Classe FileManager

I metodi sono ReadText e WriteText, entrambi dediti alla creazione del file XML che racchiude le proprietà dell'oggetto Query grafico.

Metodo	Descrizione
findSub	Ricerca all'interno della stringa una query annidata; se la trova la prende e la memorizza in una coda.
parseSub	Sostituisce in luogo della query annidata una carattere speciale identificativo: ###subquery###. Questo per evitare di analizzarla assieme al resto della stringa, che potrebbe causare problemi.
splitQuery	Analizza la stringa SQL originale e la divide in base ai suoi principali elementi costitutivi in un array di 6 elementi.

Figura 4.10: Metodi della classe Analyze

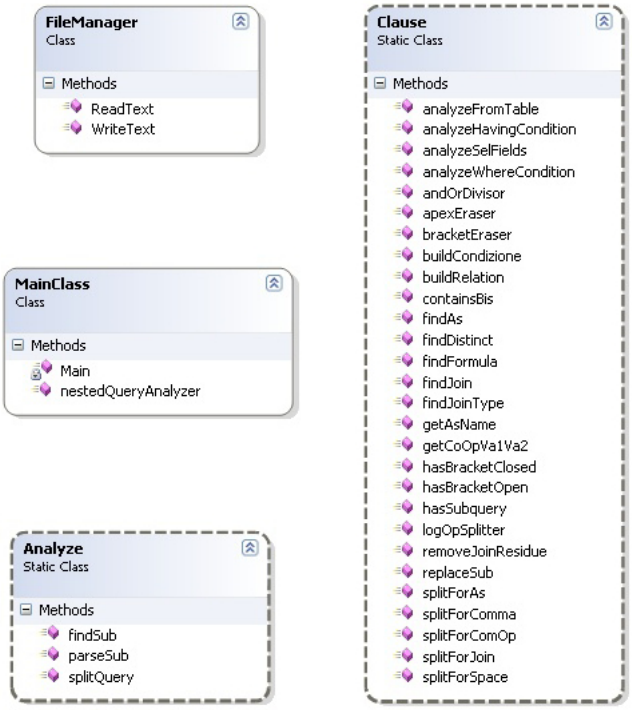

Figura 4.11: Struttura delle classi del progetto

4.6 Funzionalità

Per avere un'idea visiva di cosa faccia effettivamente il progetto, ne viene qui proposto un esempio.
Inizialmente, dall'ambiente grafico di rappresentazione della query si poteva passare alla visualizzazione dello script SQL relativo, ma non si poteva effettuare il passaggio inverso poiché mancava l'implementazione di un motore che analizzasse lo script della query e la ricostruisse in veste grafica.

Ora, dalla modalità grafica della query di figura 4.14 cliccando il tasto sql si passa alla modalità codice mostrata in figura 4.15; mentre prima non era possibile eseguirlo ora, cliccando sul pulsante Grafico, si può ricostruire la query a livello

Metodo	Descrizione
analyzeFromTable	Uno dei 4 metodi principali, è quello che si occupa dell'analisi della stringa relativa alla clausola FROM; scompone, attraverso metodi accessori, gli elementi costitutivi del codice e ricostruisce le varie tabelle con le loro relative proprietà all'interno dell'oggetto Query.
analyzeHavingCondition	Uno dei 4 metodi principali, si occupa dell'analisi dell'eventuale clausola HAVING. Il suo funzionamento è identico a quello di analyzeWhere condition.
analyzeSelFields	Uno dei 4 metodi principali, primo ad essere chiamato, è responsabile della suddivisione dei campi della stringa SELECT, della loro analisi e della loro ricostruzione all'interno dell'oggetto Query.
analyzeWhereCondition	Uno dei 4 metodi principali, scompone e analizza la stringa relativa alla clausola WHERE, effettuando in seguito la ricostruzione delle condizioni elencate della stringa SQL ed inserendole nell'oggetto Query.
andOrDivisor	Divide i campi della stringa usando "AND" e "OR" come separatori. Gli elementi verranno memorizzati uno ad uno all'interno di una coda. (Usato in caso di operatore di condizione "ON" nella clausola FROM per una determinata relazione)
apexEraser	Rimuove le virgolette (") dalla stringa.
bracketEraser	Rimuove eventuali "(" e ")" dai campi analizzati.
buildCondizione	Costruttore per una condizione da inserire nella WHERE o nella HAVING.
buildRelation	Costruttore per una relazione di JOIN tra tabelle da inserire nella FROM.
containsBis	Metodo per controllare che una stringa ne contenga un'altra; è usato per valutare gli elementi di ORDER BY e GROUP BY.
findAs	Ricerca e segnala la presenza di un eventuale operatore di alias.
findDistinct	Ricerca la presenza dell'operatore SELECT DISTINCT all'interno della clausola SELECT.
findFormula	Segnala o meno la presenza di una funzione all'interno di un campo della clausola SELECT.
findJoin	Ricerca dell'operatore di Join (di qualunque tipo) all'interno della clausola FROM.

Figura 4.12: Metodi della classe Clause

findJoinType	Se è presente un operatore di Join, questo metodo ricerca di che tipo è l'operatore trovato.
getAsName	Metodo che dato un elemento, ne ricerca l'alias associato.
getCoOpVa1Va2	Presa una condizione, ne individua, scompone ed analizza colonna, tipo di operatore, valore nr1 e valore nr2.
hasBracketClosed	Ricerca e segnala la presenza o meno del simbolo ")" alla fine della stringa.
hasBracketOpen	Ricerca e segnala la presenza o meno del simbolo "(" all'inizio della stringa.
hasSubquery	Ricerca e segnala la presenza di una eventuale query annidata cercando il carattere speciale ###subquery### all'interno della stringa.
logOpSplitter	Divide i campi della stringa usando "AND" e "OR" come separatori. Gli elementi verranno memorizzati uno ad uno all'interno di una coda. (Usato per separare le condizioni della clausola WHERE o HAVING)
removeJoinResidue	Rimuove residui dell'operatore di JOIN nelle stringhe delle tabelle da analizzare.
replaceSub	Usato nelle condizioni WHERE, rimpiazza il codice ###subquery### con la relativa query annidata.
splitForAs	Divide gli elementi della stringa usando "AS" come separatore.
splitForComma	Divide gli elementi della stringa usando la virgola "," come separatore.
splitForComOp	Divide gli elementi della stringa usando gli operatori di comparazione come separatori.
splitForJoin	Divide gli elementi della stringa usando "JOIN" come separatore.
splitForSpace	Divide gli elementi della stringa usando lo spazio " " come separatore.

Figura 4.13: Metodi della classe Clause

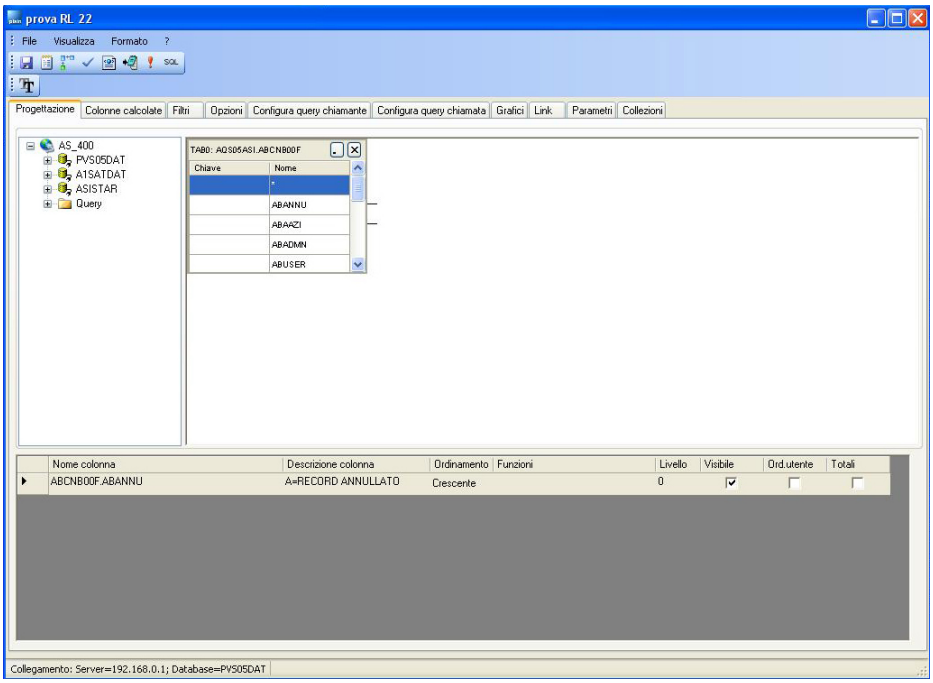

Figura 4.14: Modalità grafica di partenza

grafico come evidenziato in figura 4.16.
E' possibile inoltre effettuare modifiche allo script SQL: in tal modo, il sistema registrerà le modifiche e opererà i corretti cambiamenti alla query grafica.
Aggiungendo alla query di prova una clausola WHERE come si vede in figura 4.17 e ricostruendola graficamente, si nota non solo che la query è stata comunque ricostruita (figura 4.18) ma anche che, nel separatore Filtri, è stata valutata l'aggiunta della clausola WHERE e come tale è stata valorizzata correttamente (figura 4.19).

A livello di codice, sono stati effettuati i test di integrità su tutti i metodi con valore di ritorno (eccetto nestedQueryAnalyzer poiché il valore di ritorno è un valore casuale, e pertanto non prevedibile a priori) di tutte le classi implementate. Tutti i test hanno dato esito positivo.
Empiricamente, sono stati fatti circa duecento test su query sia prese dal database aziendale sia query inventate o copiate da esempi pratici. Molti di questi test sono stati fatti in fase di sviluppo, per vederne il comportamento e prevenire o correggerne l'errato funzionamento. Spesso si è insistito anche su esempi di query molto complesse, per far si che il prodotto non sia impreparato ai casi peggiori che possa affrontare.
A livello di inserimento da console o tramite la lettura di un file di testo, le query

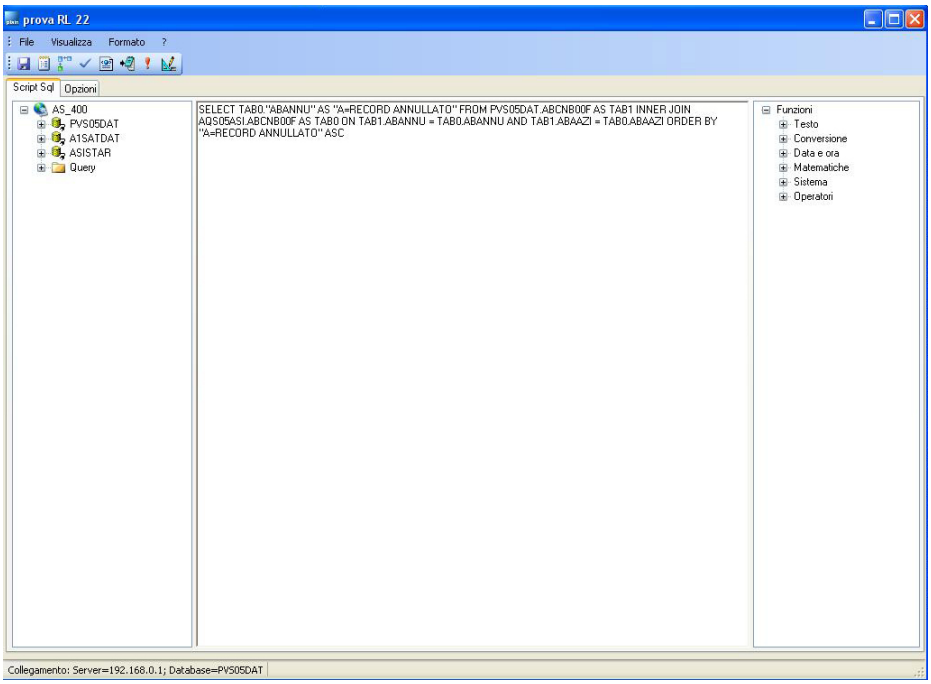

Figura 4.15: Modalità script SQL

analizzate sono state tutte correttamente ricostruite entro le funzionalità ed i limiti sopra descritti. La verifica del file XML nei suoi minimi dettagli ed il confronto con la stringa di partenza sono stati il metro di giudizio utilizzato.

Dopo l'implementazione con la suite aziendale, le query rispettanti i requisiti necessari sono state correttamente riconvertite e, se modificate, hanno reagito positivamente alla costruzione.

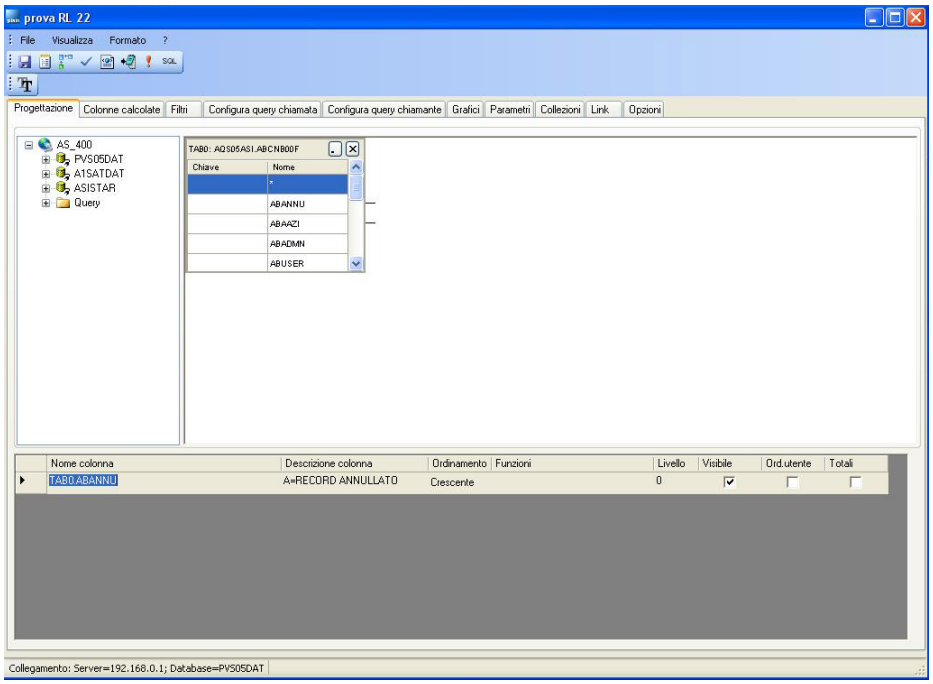

Figura 4.16: Modalità grafica ricostruita

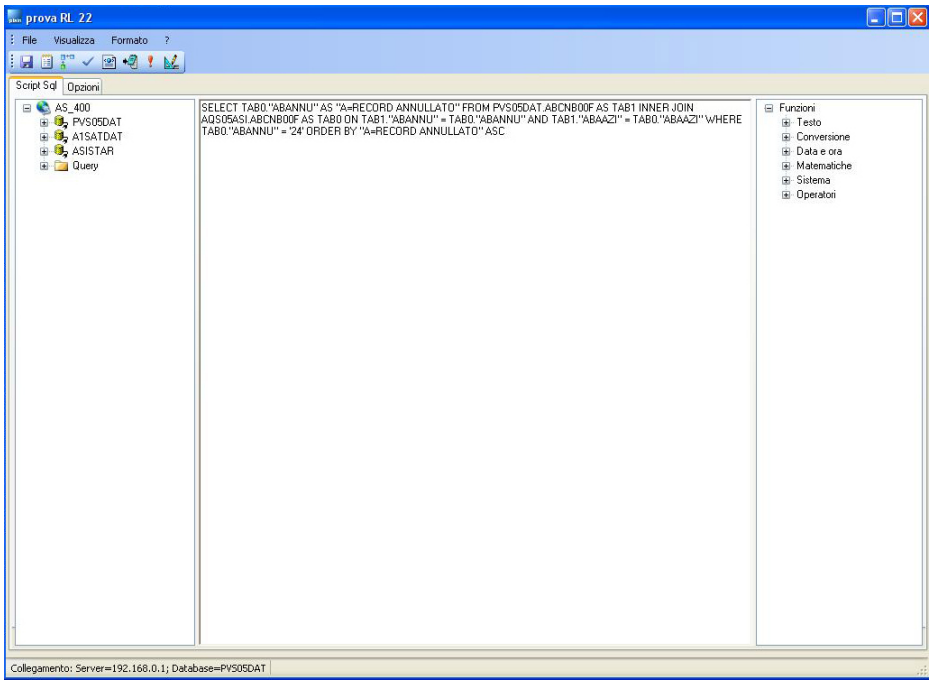

Figura 4.17: Aggiunta della condizione WHERE allo script SQL

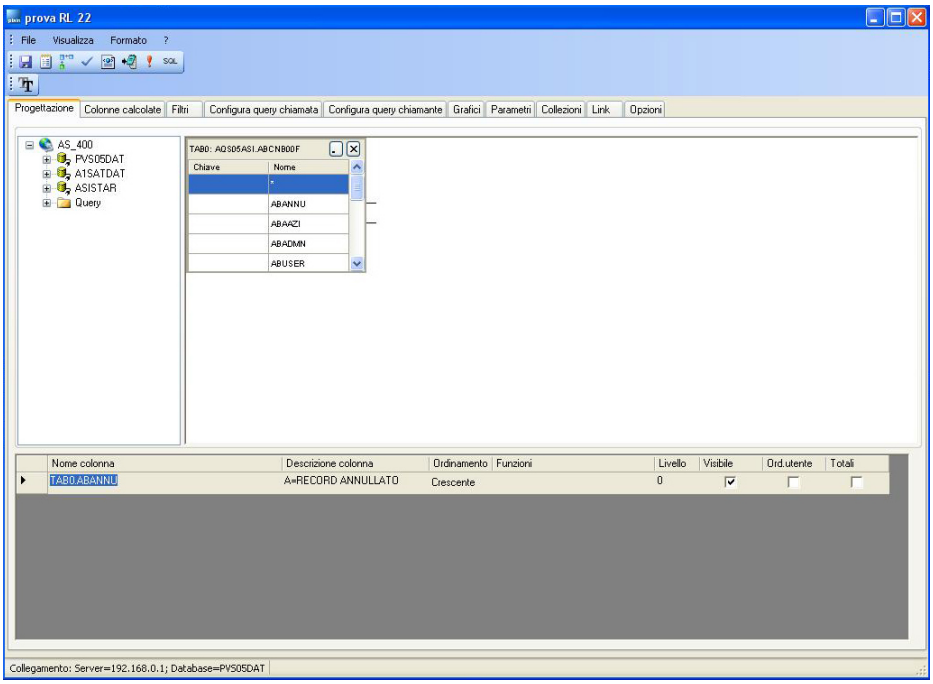

Figura 4.18: Modalità grafica con nuova condizione

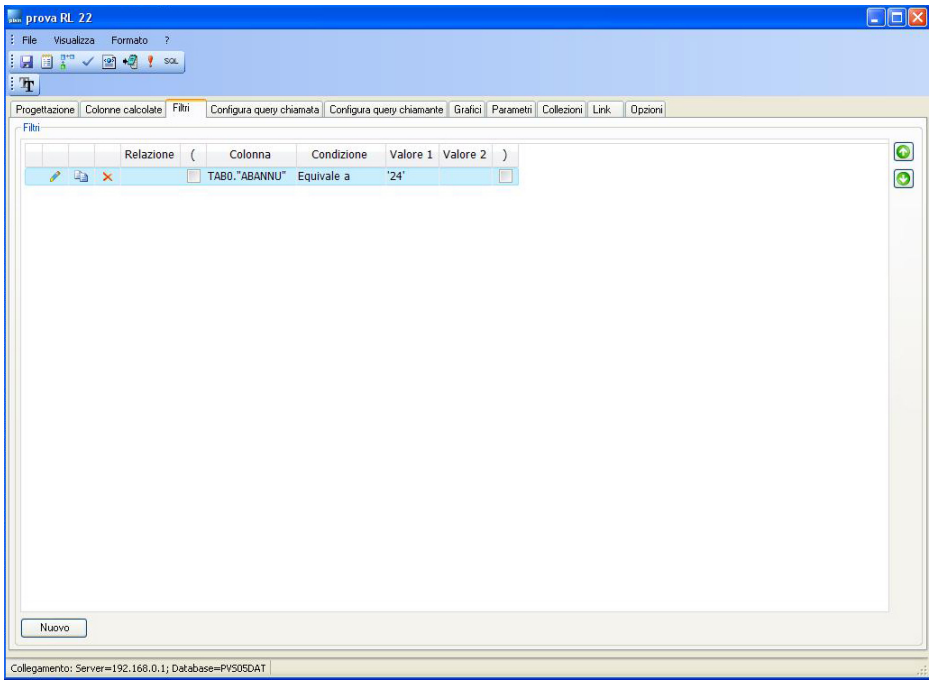

Figura 4.19: Verifica della presenza della nuova condizione

Capitolo 5

Ulteriori Sviluppi

Durante lo studio e l'implementazione dell'algoritmo, si sono presentate occasioni in cui poter ampliare le funzionalità del progetto oltre a quelle richieste dalle specifiche. Molte di queste funzionalità non trovano per ora applicazione in quanto non supportate dal DDPv2 e dall'EDP.
Si gettano le basi per uno sviluppo futuro di tali applicazioni, migliorando e arricchendo il patrimonio di funzioni implementate dai programmi della suite plain[1].
La possibilità di riconoscere le parole chiavi degli altri tipi di comandi SQL permetterà una prima individuazione sintattica del costrutto di tali operazioni.
Le query annidate e quelle con operatori insiemistici sono per ora non costruibili dal DDPv2 nonostante l'algoritmo riesca a riconoscerle ed a scomporle: implementando tale funzione nell'EDP sarebbe possibile estenderne le capacità di costruzione grafica, visualizzando risultati non ottenibili prima. Molti operatori inoltre, seppur facenti parte del linguaggio standard SQL 92, non trovano applicazione nel DDPv2, poiché non sono stati definiti a priori: a causa di tale mancanza ne è impossibile una loro valutazione e costruzione. I tipi di JOIN implementati sono inferiori a quelli esistenti, ed alcune funzioni della clausola SELECT, quali DISTINCT e ALL, non possono esser utilizzati in quanto non definiti nel codice della suite aziendale.
L'algoritmo è in grado di riconoscere tali tipi di operatori e di valorizzarli correttamente, nel caso un giorno anche il DDPv2 e l'EDP riescano a farlo.

[1] http://www.plain.it/

5.1 Individuazione di altri comandi SQL

Nonostante le uniche stringhe SQL analizzabili dal progetto dovessero essere le query, ci fu una richiesta di sviluppare un piccolo applicativo per la verifica del tipo di stringa: a seconda dell'operatore contenutovi, avrebbe potuto essere una query o un altro comando eseguibile sulla struttura del database.
Per verificarne il tipo di appartenenza, è stato costruito un elenco di regular expression ognuna delle quali identificava nella stringa un particolare tipo di comando, quali ad esempio INSERT, UPDATE, DELETE, CREATE, DROP, ALTER e altri.
In base a tali tipi di espressioni è facile individuare la tipologia del comando; una volta riconosciuta, viene memorizzata all'interno di una variabile di tipo String e comunicata all'utente tramite la finestra di terminale.
Questa implementazione, seppure inutilizzabile dal DDPv2 in quanto opera solo con stringhe rappresentanti query, permette di poter riconoscere i diversi tipi di comandi, preparando le basi per l'analisi anche di queste stringhe di codice SQL.

5.2 Analisi di query annidate

Dalle specifiche di progetto, le query annidate qualora fossero state presenti nel codice dovevano essere riconosciute, memorizzate come nome di una tabella e non analizzate dall'algoritmo. Questo poiché l'EDP non è ad oggi in grado di poterle costruire graficamente.
All'interno del progetto invece è stata implementata la possibilità di analizzarle: una volta individuate e tolte dal codice principale, vengono memorizzate in una coda di elementi di tipo String e, all'occorrenza, vengono richiamate ed analizzate come fossero una query di primo livello. Ciò consente la creazione di un oggetto di tipo Query, di cui ne è possibile una rappresentazione grafica.
Il limite che si presenta è quello dovuto al DDPv2: non essendo in grado di utilizzare un oggetto di tipo Query come parametro per una tabella, non è in grado di collegare i risultati e di rappresentarli graficamente.
Nel caso in cui si apportassero delle modifiche alla classe DDPv2 e si riuscisse ad inserire un oggetto Query come parametro di un altro oggetto Query, l'algoritmo sarebbe già pronto a produrne il codice.
Una alternativa sarebbe quella di generare il risultato della query annidata, di trasformarlo in un tipo di dati Tabella e di poterlo inserire separatamente come parametro all'interno della query di primo livello: aggiungendo un paio di righe

di codice miranti alla trasformazione da tipo di dato Query a Tabella il progetto si troverebbe pronto all'esecuzione.

5.3 Individuazione di query insiemistiche

Il DDPv2 è in grado di costruire query a livello grafico, ma non è in grado di rappresentare graficamente le operazioni insiemistiche; per questo non è stata richiesta nelle specifiche l'analisi di tali operatori. Nel caso si trovi una UNION o un operatore simile, l'algoritmo deve limitarsi a riconoscerlo, scomporre la stringa nelle due query collegate ed analizzare solo la prima.
QueryAnalyzer e LibQueAnalyzer sono in grado di analizzare anche la seconda stringa e di riconoscere l'operatore utilizzato per il collegamento delle query: riescono quindi a valorizzare gli elementi di una query con operatori insiemistici. La limitazione posta è quindi da imputarsi al DDPv2 che non appena riuscirà a costruire graficamente gli operatori insiemistici sarà in grado di sfruttare a pieno questa potenzialità dell'algoritmo.

5.4 Estensione del dizionario per i tipi di JOIN

Ogni volta che si crea una relazione tra le tabelle attraverso l'operatore di Join, all'interno dell'oggetto Query tale relazione viene identificata da un carattere univoco che indica il tipo di JOIN che sussiste tra le tabelle.
Ad ora, il DDPv2 è in grado di riconoscere correttamente solo sei tipi di JOIN, due dei quali non implementati nello standard SQL 92. Il progetto, invece, riesce a riconoscere ed a distinguere tutti i tipi di JOIN definiti nello standard SQL 92, associando ciascuno di essi ad altrettanti caratteri univoci e per questo distinguibili tra loro.
Il compito di individuare il tipo di JOIN utilizzato spetta al metodo findJoinType, della classe Clause del progetto realizzato: tale metodo riceve come parametro di ingresso una variabile s di tipo String contenente il segmento di codice in cui si vuole individuare il tipo di JOIN. Viene creato un elenco di regular expression, una per ogni tipo di JOIN definito nel linguaggio SQL standard 92 a cui vanno aggiunti i tipi personali di JOIN che utilizza l'azienda: ognuna di queste viene valutata per vedere se è presente all'interno della stringa. Ad ogni tipo di JOIN è inoltre associato un carattere di tipo Char che indicherà all'interno dell'oggetto il tipo di relazione esistente tra le tabelle. Mentre per i JOIN implementati dal

Tipo di JOIN	Codice identificativo	Implementato da DDPv2
NATURAL JOIN	N	NO
CROSS JOIN	C	NO
FULL JOIN	A	NO
SELF JOIN	S	NO
UNION JOIN	U	NO
RIGHT JOIN	R	SI
LEFT JOIN	L	SI
INNER JOIN	I	SI
LEFT INNER JOIN	G	NO
RIGHT INNER JOIN	J	NO
FULL INNER JOIN	B	NO
OUTER JOIN	O	NO
LEFT OUTER JOIN	H	NO
RIGHT OUTER JOIN	K	NO
FULL OUTER JOIN	F	SI

Figura 5.1: Tipi di JOIN analizzati

DDPv2 era già stato assegnato il valore Char, per tutti gli altri tipi di JOIN i Char relativi sono stati assegnati in fase di creazione dell'algoritmo, cosicché ogni tipo di JOIN ne abbia uno.
Una volta individuato il tipo di legame, il Char relativo viene registrato all'interno della variabile code e restituito al metodo chiamante; tale valore verrà poi utilizzato per la creazione degli oggetti di tipo Relazione. Il DDPv2, potendo gestire solo alcuni valori Char associati al tip odi JOIN, non sarà in grado di costruire relazioni con valori diversi a quelli del suo dizionario.
Ampliando il dizionario del DDPv2 seguendo le linee guida definite in questo progetto, sarà possibile sfruttare la potenzialità di riconoscimento di tutti i tipi di JOIN qui implementata.

5.5 Individuazione di altri operatori

Nella clausola SELECT è possibile utilizzare operatori particolari quali *, equivalente ad un SELECT ALL, che seleziona i campi di tutte le tabelle coinvolte nel risultato, e DISTINCT che serve ad eliminare dalle tabelle risultato record con gli stessi valori.
Entrambi questi operatori non vengono riconosciuti dal DDPv2, mentre l'algoritmo è in grado di riconoscerli; si comporta in modo diverso a seconda di quale dei due trova all'interno della clausola:

- Nel caso in cui individui la parola chiave * oppure ALL, non essendo in grado di valutarla, termina il programma restituendo sulla finestra di terminale una frase che ne indica la presenza all'interno della query e che per questo ne termina l'esecuzione;
- Se individua la parola chiave DISTINCT, non essendo in grado di valutarlo, continua l'esecuzione del processo ma non eliminerà i risultati doppi. L'utente sarà avvertito di tale fatto sempre tramite una frase sulla finestra di terminale.

Poiché il progetto prevede il riconoscimento di tali operatori, implementandone la valutazione e la creazione di tali funzionalità all'interno del DDPv2 si riuscirebbe a sfruttare tale parte del codice per ora non utilizzata.

Capitolo 6

Conclusioni

Lo scopo dello stage è stato quello di sviluppare un algoritmo per la scomposizione ed il riconoscimento delle componenti di una query in codice SQL.
La durata dello stage di 250 ore è stata più che sufficiente per sviluppare il progetto in tutte le sue fasi, lasciando anche del tempo per implementazioni aggiuntive utili in futuro.
Nella prima fase di formazione si è posta la concentrazione sulla filosofia di riconoscimento delle stringhe SQL; effettuando ricerche su internet e su testi riguardanti tale argomento non è stato trovato nulla in merito, poiché nessuno fino ad ora ha pubblicato o studiato una soluzione al tipo d'esigenza dell'azienda.
L'idea che sta alla base dell'algoritmo è maturata studiando la sintassi SQL e valutando centinaia di esempi di query aziendali, usate come stringhe di test in fase di realizzazione.
Oltre alla formazione sul linguaggio SQL, per la scrittura del codice, si è dovuta svolgere una fase di formazione sul linguaggio C#; successivamente si è passato alla fase operativa di costruzione dell'algoritmo. I primi passi verso la scomposizione della query nelle sue clausole è stata più volte approfondita per sperimentare diverse soluzioni in merito: l'uso delle espressioni regolari si è dimostrata la scelta migliore data la semplicità costruttiva. Su questo si sono poi basate in generale le scomposizioni seguenti. In corso d'opera si ha avuto la possibilità di implementare delle funzionalità non richieste dall'azienda; il volerle comunque creare è nato dal desiderio sia di voler potenziare l'algoritmo per renderlo versatile anche su altre piattaforme aziendali, sia poiché erano facilmente inseribili nel codice. Inoltre, dovendo rispettare l'SQL standard 92, è sembrato giusto dover implementare la possibilità di analizzare quei costrutti che, anche se non utilizzati dall'EDP, sono definiti nel linguaggio.
L'analisi di operatori aggiuntivi, ad esempio, è stato implementando semplice-

mente aggiungendo delle espressioni regolari con poche righe di codice.
Una volta terminata la prima stesura del codice è stata effettuata una revisione completa di tutte le sue parti, migliorando alcuni cicli di ricerca nel pattern matching, ove richiesto, e nell'utilizzo delle strutture dati.
Sono stati effettuati numerosi test con un sottoinsieme delle query aziendali per valutare la correttezza nella creazione dell'oggetto di tipo Query: confrontando il codice prodotto in XML si è visto che l'algoritmo realizzava correttamente la ricostruzione della query. Ulteriori test sono stati effettuati collegando l'algoritmo direttamente al database aziendale, da cui si sono reperite nuove query per valutare la correttezza dell'algoritmo anche su stringhe di codice SQL mai viste sino ad allora: i risultati sono stati più che soddisfacenti.
Per completare l'opera, si è inserita la libreria contenente l'algoritmo all'interno dell'EDP per verificare la compatibilità del progetto con le specifiche fornite dall'azienda. Questo passaggio ha presentato notevoli difficoltà: essendo scritto in linguaggio Visual Basic si è prima dovuto conoscere le nozioni fondamentali di tale linguaggio e poi procedere alla lettura del codice per individuare i punti in cui l'EDP bloccava la ricostruzione grafica della query. Sono stati effettuati reindirizzamenti di metodi, modifica di variabili, modifica di alcuni spezzoni di codice e diversi test per verificarne la funzionalità.
Questi test hanno messo in luce alcune limitazioni rispetto alle potenzialità che il progetto offre. In figura 6.1 sono state evidenziate tali limitazioni, da cosa sono dovute e una possibile soluzione.

Alcune di queste limitazioni sono poste dai software aziendali plain: l'impossibilità di valutazione di certi operatori, quali tipi di JOIN, operatori insiemistici, SELECT ALL, DISTINCT e query annidate deriva dalla mancata implementazione di tali costrutti nella definizione delle classi EDP e DDPv2.
Sebbene quindi il progetto sia in grado di andare al di la delle funzionalità richieste in fase di sviluppo, queste restano per ora inutilizzate. In futuro, nel caso si volesse implementare la possibilità di gestire gli operatori ora non riconosciuti, il grosso del lavoro andrebbe fatto nel DDPv2 e nell'EDP, mentre l'algoritmo sarebbe già pronto all'analisi.
Il lavoro svolto ha suscitato l'interesse dell'azienda, soddisfatta di essere ora in grado di arricchire l'EDP della funzionalità di ricostruzione grafica di una query a partire dal suo codice SQL. L'algoritmo inoltre risulta abbastanza portabile: cambiando i metodi per la costruzione dell'oggetto di tipo Query, l'universalità del codice SQL standard 92 riflette sul progetto la possibilità di implementarlo all'interno di altri software che necessitano tale analisi.
Con opportune modifiche, si potrebbe anche valutare un eventuale scopo didatti-

Problema	Cause	Soluzioni
L'algoritmo non è in grado di ricostruire ed analizzare query sintatticamente scorrette.	Il progetto si basa sull'analisi di query che rispettino la sintassi del codice SQL 92.	Poiché sarebbe illogico cercare di rimediare agli errori della query scorretta, si arresta il programma inviando un messaggio alla finestra di terminare con la spiegazione della terminazione.
L'algoritmo non è in grado di analizzare e ricostruire comandi di aggiornamento, inserimento ed eliminazione.	Il progetto è stato realizzato allo scopo di valutare esclusivamente stringhe contenenti query SQL.	Essendo stato realizzato a parte una classe per l'individuazione del tipo di comandi, si potrebbe creare un'analisi algoritmicamente simile a quella qui proposta per estenderne la possibilità. Ciò tuttavia non basterebbe: anche l'EDP e il DDPv2 devono implementare classi e metodi atti all'analisi di tali comandi.
L'algoritmo non è in grado di analizzare e valorizzare correttamente operatori non appartenenti al linguaggio SQL standard 92.	Le specifiche di progetto richiedevano l'inserimento nel dizionario di operatori appartenenti all'SQL standard 92 e di operatori utilizzati dall'azienda.	Basta estendere il dizionario del progetto con l'aggiunta di espressioni regolari mirate all'individuazione di operatori non implementati.
Alcune query con all'interno query annidate non vengono correttamente convertite.	1. In fase di passaggio da ambiente grafico a script SQL nell'EDP la stringa di codice SQL non viene correttamente tradotta comportando la presenza di errori sintattici che inducono il progetto ad effettuare una analisi errata. 2. Non vengono ricostruite correttamente a livello grafico poiché il DDPv2 non è in grado di costruire le query annidate.	1. Migliorare l'algoritmo di traduzione dell'EDP affinché traduca correttamente il codice di query multilivello nel passaggio da rappresentazione grafica a script SQL. 2. Implementare nel DDPv2 la funzione che permette di costruire graficamente il risultato di query annidate; nel progetto è stata simulata una soluzione a questo problema, facendo risultare l'algoritmo pronto a sviluppare tale funzionalità.
Query con un elevato numero di tabelle da visualizzare hanno una errata visualizzazione grafica.	Errore dell'EDP nella costruzione grafica del codice SQL.	Migliorare l'implementazione per la visualizzazione grafica nell'EDP.
Mancato inserimento di apici per gli identificatori di tabelle e colonne (soprattutto nella clausola FROM)	Errore dell'EDP nella costruzione del codice della query in SQL.	Migliorare l'implementazione per la costruzione del codice SQL della query nell'EDP.

Figura 6.1: Limitazioni, cause e possibili soluzioni

co del progetto: nello studio dei fondamenti del linguaggio SQL, sarebbe possibile evidenziare i passaggi costruttivi di una query, delle sue clausole e dei suoi elementi più semplici.

Bibliografia

1. R.A. Elmasri, S.B. Navathe, Sistemi di basi di dati - Fondamenti,Pearson Addison Wesley (2007)
2. C. Gross, Beginnign C# - From Novice to Professional, Apress (2008)
3. D. Solis, Illustrated C#, Apress (2008)
4. J. Hilyard, S. Teilhet, C# Cookbook, O'Reilly (2007)
5. J. Bishop, C# 3.0 Design Patterns, O'Reilly (2007)
6. J.E.F. Friedl, Mastering Regular Expression 3rd edition, O'Reilly (2006)
7. J. Liberty, D. Xie, Programming C# 3.0, O'Reilly (2007)
8. A. Troelsen, Pro C# and the .NET 3.5 Platform, Apress (2007)
9. J. Maddock, Boost.Regex (2007)
10. M.T. Goodrich, R. Tamassia, Data Structures and Algorithm in Java, John Wiley & Son (2007)
11. Chamberlin, Astrahan, System R, IBM corp. (1981)
12. G. Gradenigo, Appunti del corso di Basi di Dati (2008)
13. I. Bartolini, Appunti dal corso di Sistemi Informativi (2009)
14. http://www.sql.org/
15. http://www.msdn.microsoft.com/
16. http://www.w3schools.com/sql/
17. http://www.sqlcourse.com/
18. http://www.contrib.andrew.cmu.edu/ shadow/sql/sql1992.txt

19. http://www.wikipedia.org/

20. http://www.plain.it/

21. http://www.hwupgrade.it/

22. http://www.csharp-station.com/

23. http://www.radsoftware.com.au/

24. http://www.dotnetarchitects.it/

25. http://www.regular-expressions.info/