Università degli Studi *"Roma Tre"*
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Studio di metodologie di ricerca su repository mediante Ferret, motore di indicizzazione e ricerca

Progetto complementare per il corso "Complementi di Basi di Dati"

Luca Lanziani
mat. 289149

Federico Spini
mat. 240917

Anno Accademico 2008/2009

Obiettivo

Il nostro lavoro ha sondato la possibilità di rappresentare i dati contenuti in un repository (progetto GAIA) in modo che fossero indicizzabili e ricercabili mediante un motore di indicizzazione e ricerca testuale quale ad esempio Lucene. Nella fattispecie è stato adoperato il linguaggio Ruby e il motore ad esso correlato: *Ferret* . Lo sforzo maggiore, oltre alla buona quantità di codice generato, è stato speso nel cercare una codifica testuale delle informazioni contenute nel repository tale che fosse possibile crearvi sopra un indice di cerca proprio mediante *Ferret* ed effettuarvi ricerche. Tali ricerche, proprio grazie alla tecnologia del motore di ricerca adottato dovrebbero risultare estremamente più rapide, che se non effettuate su database sfruttando una tecnologia basata essenzialmente su SQL.

Il modello dell'indice

Il modello di indice di ricerca sfruttato da *Ferret* prevede la definizione del *Document* , l'elemento minimo di indicizzazione e di ricerca. Ogni *Document* è strutturato in *Field* , in ognuno dei quali possiamo andare ad inserire stringhe utili all'indicizzazione. Ovviamente la struttura utilizzata da *Ferret* nella sua interezza è ben più complessa. Quelli citati sono solo gli elementi utili ai nostri scopi.

Strutturazione dell'indice

Di seguito si illustra la struttura dell'indice di ricerca generato a partire dai dati del repository, la conoscenza della cui struttura è data per scontata.

Scelta del *Document*

Il *Document* coincide con uno Schema Exchange (di seguito `SE`). Grazie a tale scelta con un solo scorrimento dell'indice si riescono a trovare gli Schema Exchange che corrispondono ai criteri di ricerca desiderati.

Composizione dei *Field*

Un *Document* è composto nel modello presentato dai seguenti *Field* :

:n_rel → Numero delle relazioni presenti nello `SE` ;

:n_tot_key → Numero totale delle chiavi nello `SE` ;

:n_tot_fkey → Numero totale delle chiavi esterne nello `SE` ;

:cod_rels → Stringa contenente le codifiche testuali (basate sulle proprietà) delle relazioni nello `SE` ;

:o_cod_rels → Stringa molto simile alla precedente (diversa solo per l'assenza di spazi bianchi tra le codifiche delle varie relazioni) utile per sistemare alcune questioni legate allo scoring dei risultati ottenuti;

:cod_fkey → Stringa contenente le codifiche testuali (basate sulle proprietà) dei riferimenti descritti dalle chiavi esterne delle tabelle nello `SE` .

Codifica testuale delle relazioni sulla base delle proprietà

La stringha inserita nel *Field :cod_rels* è composta dalla concatenazione di codifiche testuali delle relazioni nello `SE` separate da uno spazio. Tale codifica è ottenuta sulla base delle proprietà di ogni singola relazione: `#K#F#A`$a^{(i+1)}$. Ovvero è composta dalla concatenazione di:

- numero di chiavi presenti nella relazione (`#K`) espresse mediante due cifre in notazione decimale (cioè possono essere rappresentate al massimo 99 chiavi);
- numero di chiavi esterne presenti nella relazione (`#F`) espresse mediante due cifre in notazione decimale (cioè possono essere rappresentate al massimo 99 chiavi esterne);
- numero delle relazioni all'interno dello `SE` che hanno una chiave esterna con riferimento alla relazione in oggetto, (`#A`) espresse mediante due cifre in notazione decimale (cioè possono essere rappresentate al massimo 99 chiavi esterne);
- numero di attributi posseduti dalla relazione, espressi in notazione unaria su alfabeto $\{a\}$ ($a^{(i+1)}$) ma con un carattere a in più rispetto al numero di attributi;

Si sottolinea che il numero di cifre in notazione utilizzato per `#K`, `#F` e `#A` è parametrico ed è semplicemente modificabile.

Codifica testuale dei riferimenti descritti dalle chiavi esterne

La stringha inserita nel *Field :cod_fkey* è composta dalla concatenazione di codifiche testuali dei riferimenti descritti dalle chiavi esterne delle tabelle nello `SE` , separate da uno spazio. Le codifiche sono così composte: $R_i\_R_{k_1}\_\ldots\_R_{k_n}$, ovvero si concatena alla codifica di una relazione (come descritta precedentemente) mediante l'uso di un carattere "_", la codifica delle n relazioni verso cui essa ha un riferimento (chiave esterna). Le stringhe sono inserite secondo l'ordine lessigrafico delle R_i. Anche le n codifiche R_k sono inserite in ordine lessicografico e concatenate mediante carattere "_".

La query in FQL

Di seguito riportiamo un esempio di query che dovrebbe chiarire come possa essere effettuata la ricerca sull'indice generato con il modello descritto. *Ferret* mette a disposizione molti modelli di query, oltre ad un vero e proprio query language, chiamato Ferret Query Language (FQL). La query sottoposta al motore è effettuata proprio in FQL. Vediamo un esempio:

```
     n_rel: 3
AND
     n_tot_key: 3
AND
     n_tot_fkey: 2
AND
     (
          (
                cod_rels: 010001aa
           AND
                cod_rels: 010001aaa
           AND
                cod_rels 010200aa
           )
          OR
           (
                cod_rels: 010100aa*
           AND
```

```
                cod_rels: 010100aaa*
           AND
                cod_rels 010200aa*
           )
     )
AND
     (
          (cod_fkey: 010200aa_010001aa_010001aaa)
      OR
          (cod_fkey: 010200aa*_010001aa*_010001aaa*)
     )
```

Vediamo in dettaglio cosa si richiede specificando una query come quella precedente. Gli `SE` che matchano hanno le seguenti prorpietà:

- `n_rel: 3` → hanno 3 relazioni
- `n_tot_key: 3` → hanno in totale 3 chiavi
- `n_tot_fkey: 2` → hanno in totale 2 chiavi esterne
- `(cod_rels: 010001aa AND cod_rels: 010001aaa AND cod_rels 010200aa)` → hanno relazioni le cui proprietà sono codificate dalle stringhe riportate (match perfetto)
- `OR (cod_rels: 010100aa* AND cod_rels: 010100aaa* AND cod_rels 010200aa*)` → o che hanno almeno lo stesso numero di attributi (* è una wildcard, indica un numero qualunque di caratteri)
- `(cod_fkey: 010200aa_010001aa_010001aaa)` → presenta i riferimenti descritti dalla striga riportata (match perfetto)
- `OR (cod_fkey: 010200aa*_010001aa*_010001aaa*)` → o è possibile che abbia riferimenti analoghi a quelli dell'input ma tra relazioni che presentano un surplus di attributi

Da notare che il sistema di scoring attribuisce uno score maggiore al match perfetto perchè il match è su entrambi i termini in OR, quello perfetto e quello con wildcard.

Istruzioni di esecuzione

Per eseguire un test lanciare da terminale `ruby schema_searcher.rb`. Per configurare l'esecuzione modificare il valore delle tre costanti:

- `XML_INPUT_FILE` → file xml descrivente il DataExchange da ricercare tra le parti sinistre degli `SE` nel repository;
- `XML_SCHEMA_DIR` → la directory contenente i file xml descriventi gli `SE` da aggiungere nel repository;
- `XML_INDEX_DIR` → directory nella quale salvare l'indice,

nel file `schema_searcher.rb`.

Sviluppi futuri

Il nostro lavoro è terminato prima della modellazione della ricerca per quanto riguarda il match totale o parziale sulle costanti. L'indice modellato così come illustrato non consente di inserire un *Field* che modelli la presenza di costanti su particolari atomi di una relazione. La grana scelta (un *Document* per ogni `SE`) rende inpossibile inserire nell'indice informazioni di così fine grana. Sviluppi futuri potrebbero prevedere la creazione di un secondo indice nel quale modellare l'informazione sulle costanti e sul quale si proceda a ricercare solo dopo aver completato la ricerca sul primo indice, avendo cioè già effettuato la selezione dei risultati da presentare. La ricerca sul secondo indice ha il solo fine di far salire o scendere nel ranking uno schema a seconda del numero di costanti che esso condivide con quelle presenti nell'input della ricerca.