Eric Chu, Akanksha Baid, Xiaoyong Chai, AnHai Doan, Jeffrey Naughton

COMBINING KEYWORDS SEARCH AND FORM FOR AD HOC QUERYING OF DATABASES

Gruppo 15
Colombaretti Margherita
Mazza Valeria (Relatrice)
Tesini Tommaso

SCENARIO

- Reperire informazioni all'interno di un database relazionale presuppone la conoscenza di un linguaggio di interrogazione (SQL);

- Un utente non esperto può riscontrare difficoltà poiché:

  - non è a conoscenza di tali linguaggi, talvolta complessi;

  - è solito cercare informazioni tramite l'utilizzo di **keyword**, non adatte per ricerche strutturate.

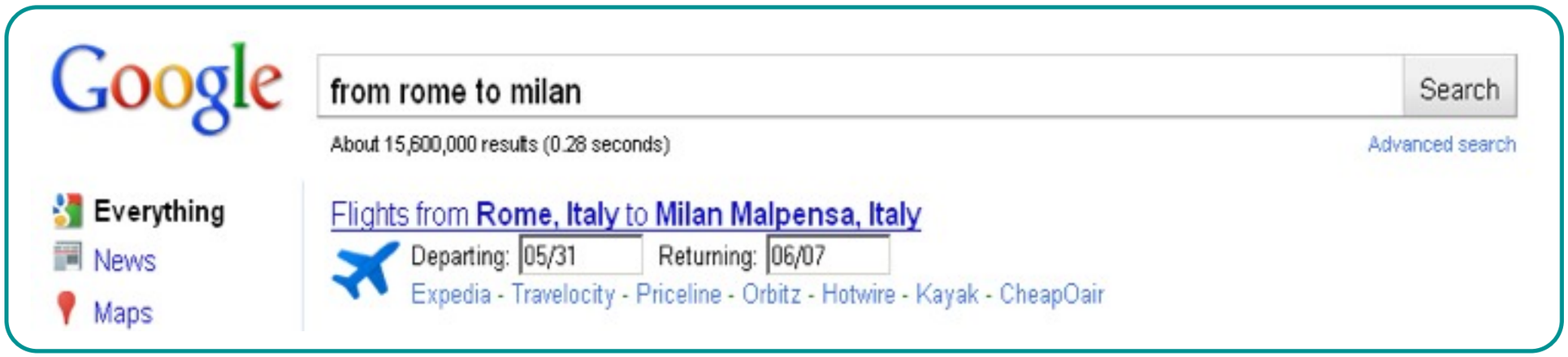
DEFINIZIONE DEL PROBLEMA

Permettere ad un utente non esperto di eseguire ricerche specifiche, integrando la facilità di utilizzo di **keyword** con la potenza di un **linguaggio di interrogazione** di database.

UN'IDEA DI SOLUZIONE...

Fornire un'intermediazione che, a partire dalle keyword, faciliti l'utente nella generazione delle query.

L'applicazione sviluppata restituisce una o più **form** che inducono l'utente a specificare dei parametri per interrogare il database in maniera strutturata.



OUTLINE

- Come creare form che soddisfino il maggior numero possibile di query?
- Quanto sono efficaci le query nell'individuare le form più attinenti?
- Come mostrare i risultati per facilitare l'individuazione da parte dell'utente delle migliori form?
- I risultati ottenuti sono incoraggianti?

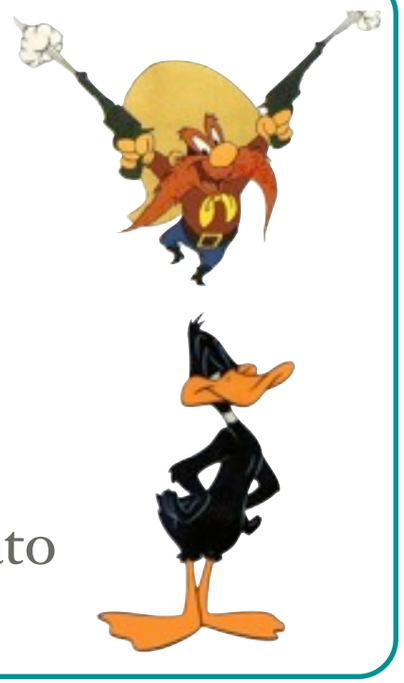
DATABASE DI ESEMPIO

TABELLE DELLE ENTITA'

personaggio (pid, nome, specie, sesso, nazione, abilità)
episodio (eid, titolo, serie, numero, anno, durata)
cartone_animato (cid, nome, creatore, anno_prima_apparizione, num_episodi)
casa_produttrice (cpid, nome)

TABELLE DELLE RELAZIONI

//associazione tra due personaggi nemici
nemici (nid, pid1,pid2)
//associazione tra cartone animato e un suo episodio
cartone_episodio (ceid, cid, eid)
//associazione tra personaggio ed episodio in cui compare
episodio_personaggio (epid, eid, pid)
//associazione tra cartone animato e casa produttrice a cui è legato
casa_cartone (ccid, cpid, cid)

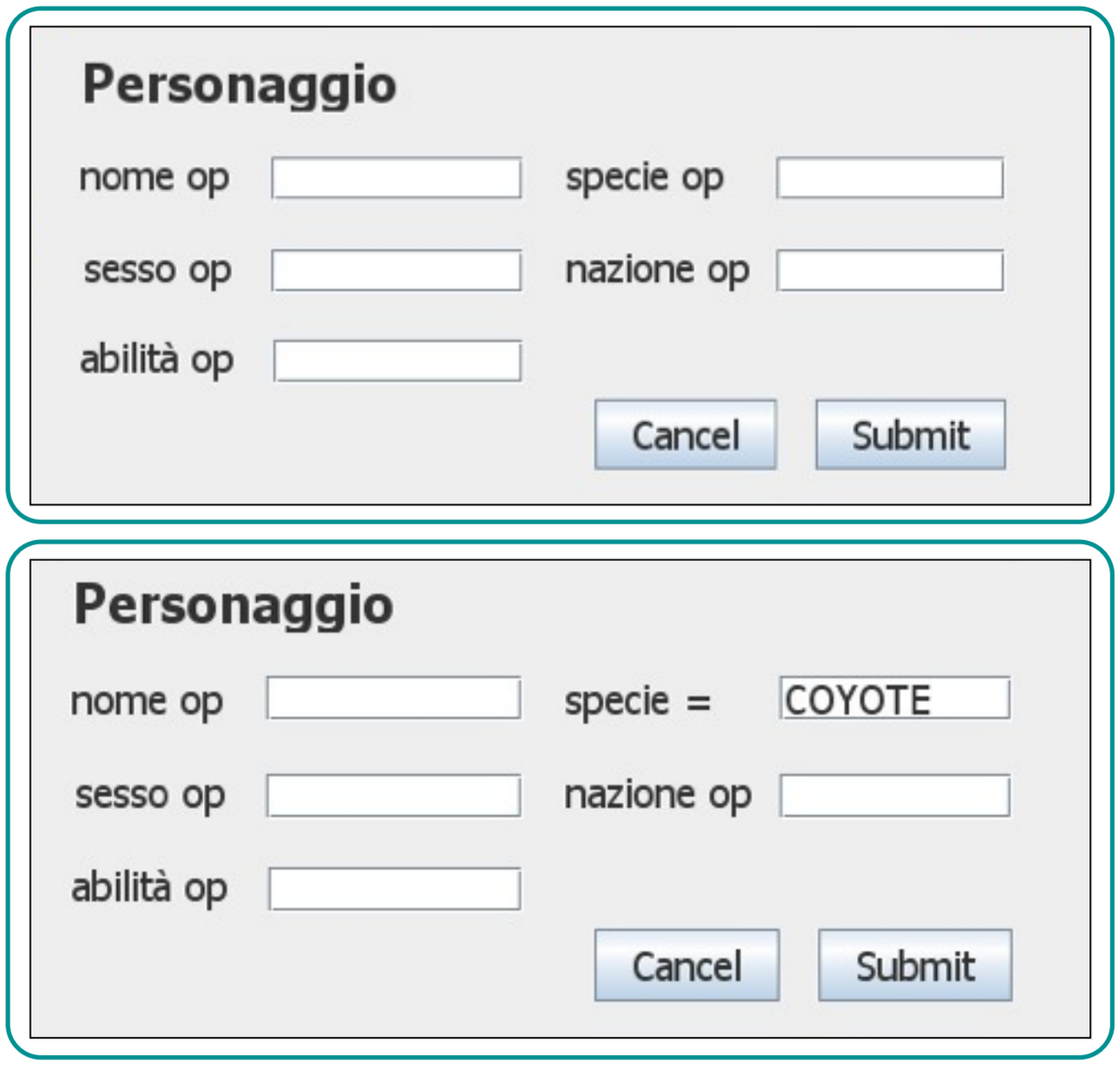


OUTLINE

- Come creare form che soddisfino il maggior numero possibile di query?
- Quanto sono efficaci le query nell'individuare le form più attinenti?
- Come mostrare i risultati per facilitare l'individuazione da parte dell'utente delle migliori form?
- I risultati ottenuti sono incoraggianti?

FORM COME TEMPLATE PER QUERY

TABELLA
personaggio (pid, nome, specie, sesso, nazione, abilità)




```
SELECT    *
FROM      personaggio
WHERE     nome op value AND
          sesso op  value AND
          abilità op value AND
          specie op value AND
          nazione op value
```



```
SELECT    *
FROM      personaggio
WHERE     specie = 'COYOTE'
```

PROCESSO DI CREAZIONE FORM

1. Specificare un sottoinsieme di SQL come linguaggio per implementare le query supportate dalle form;

2. Determinare un insieme di template “scheletri” basati sullo schema del Database;

3. Finalizzare i template modificando gli scheletri in base al livello di specificità desiderato per le form.

1. SOTTOINSIEME *SQL'*

- Il sottoinsieme di SQL , denominato SQL' , utilizzato dall'applicazione è il seguente:

SELECT (select-list)
FROM (from-list)
WHERE (qualication)
[GROUP BY (grouping-list)
HAVING (group-qualification)]

- SQL' si limita all'utilizzo di queste clausole poiché le keyword immesse dall'utente non saranno mai mappate in query eccessivamente complesse.

2. TEMPLATE "SCHELETRI" (1/2)

- L'utente non conosce lo schema del database, per questo si utilizzano template scheletri in cui vengono fissate le clausole, le relazioni e le condizioni di join e vengono parametrizzati soltanto attributi e operatori;

- Per ogni tabella delle entità E_i, viene creato il seguente template scheletro:

Ex_{basic} :
```
SELECT   *
FROM     Ei
WHERE    predicate-list
```

con $\mathbf{E_i} \in$ {personaggio, episodio, cartone_animato, casa_produttrice } e **predicate-list** contiene solo attributi che non sono chiavi.

2. TEMPLATE "SCHELETRI" (2/2)

- Nel caso in cui una tabella R_i referenzi altre tabelle con **foreign key** il suo template scheletro dovrà supportare dei **join** basati su queste chiavi, ad esempio:

EX_{fk}:

```
SELECT  *
FROM    episodio_personaggio ep, personaggio p, episodio e
WHERE   ep.pid = p.pid AND ep.eid = e.eid
        AND p.nome op expr AND... AND p.abilità op expr
        AND e. titolo op expr.... AND e. durata op expr
```

- Inoltre, vengono creati template scheletri che supportano **equijoin** su attributi che le tabelle hanno in comune ma che non sono foreign key, ad esempio:

EX_{eq}:

```
SELECT   e.titolo, ...., e.durata
FROM     episodio_personaggio ep, cartone_episodio ce, episodio e
WHERE    ep.eid = ce.eid  AND ce.eid = e.eid
         AND e. titolo op expr.... AND e. durata op expr
```

3. TEMPLATE FINALI (1/2)

La Specificità di una form si misura tramite due parametri:

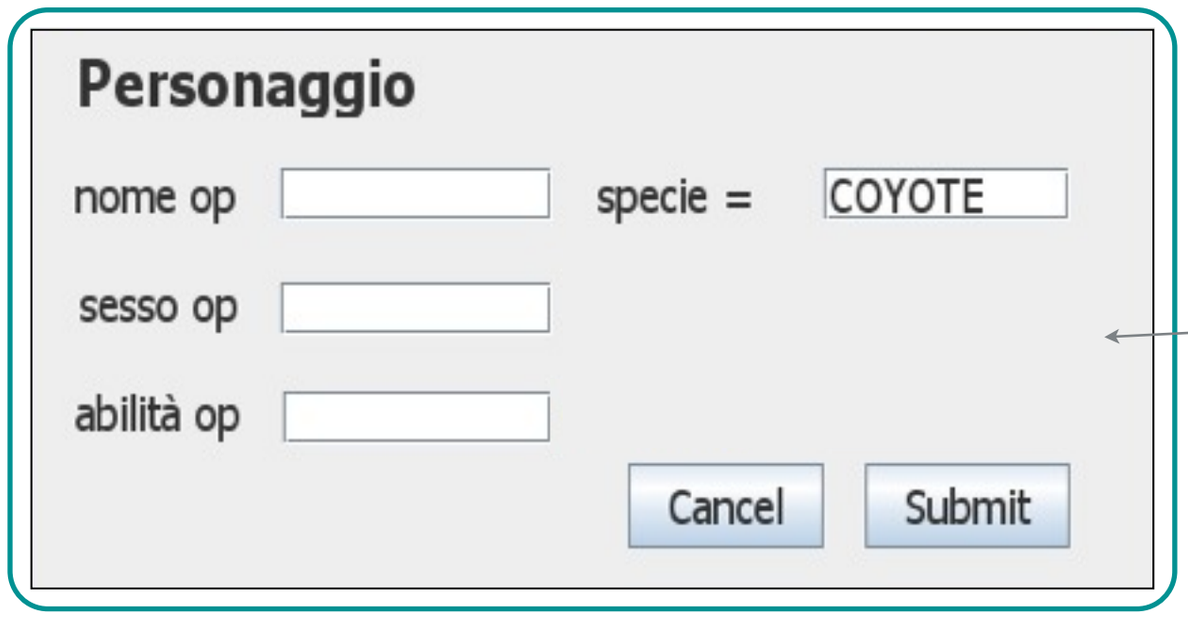
- **Form Complexity**, riguardante il numero di parametri di una form;
- **Data Specificity**, riguardante il numero di parametri di una form con valori fissati.

Aumento
Form Complexity

3. TEMPLATE FINALI (1/2)

La Specificità di una form si misura tramite due parametri:

- **Form Complexity**, riguardante il numero di parametri di una form;
- **Data Specificity**, riguardante il numero di parametri di una form con valori fissati.

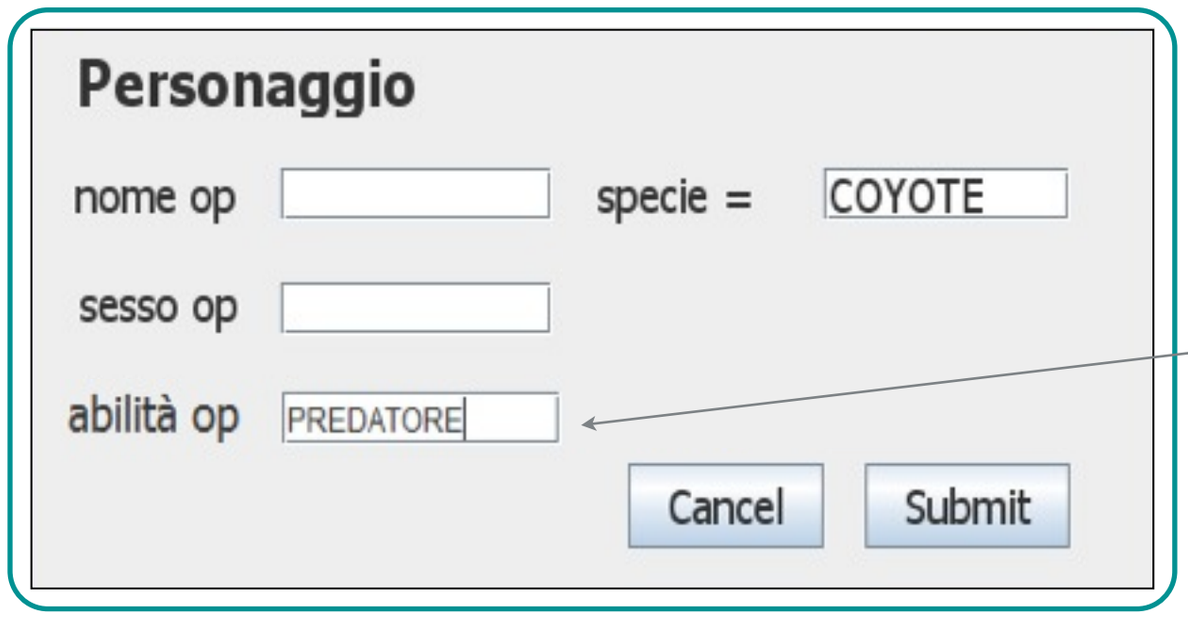


Diminuzione
Form Complexity

3. TEMPLATE FINALI (1/2)

La Specificità di una form si misura tramite due parametri:

- **Form Complexity**, riguardante il numero di parametri di una form;
- **Data Specificity**, riguardante il numero di parametri di una form con valori fissati.

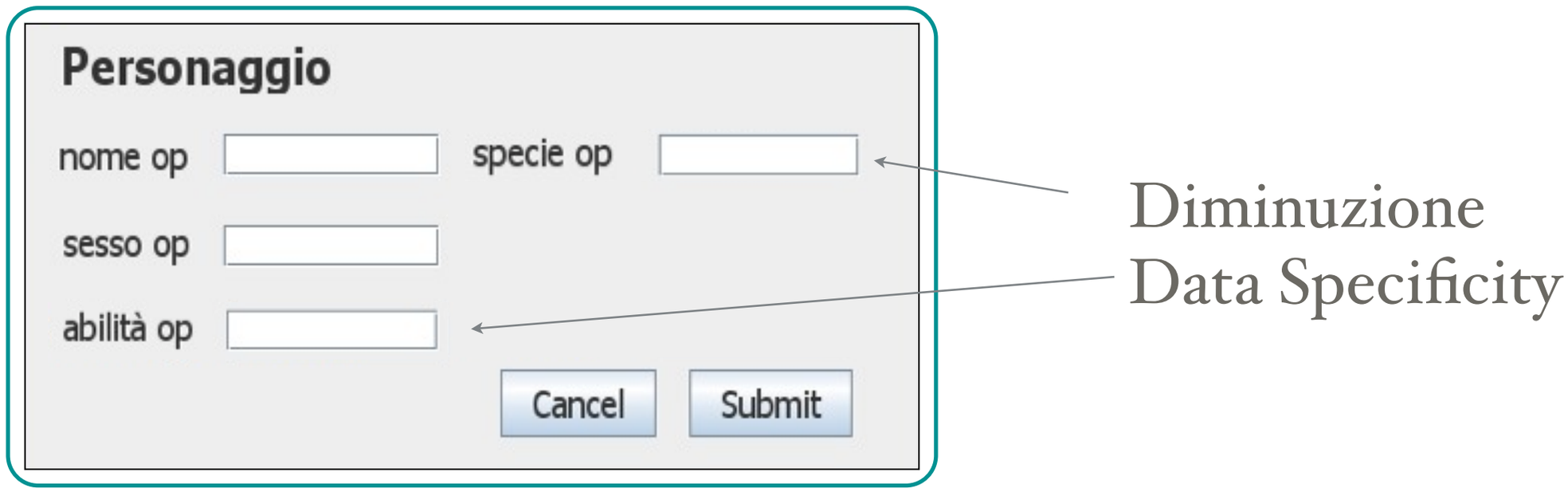


Aumento
Data Specificity

3. TEMPLATE FINALI (1/2)

La Specificità di una form si misura tramite due parametri:

- **Form Complexity**, riguardante il numero di parametri di una form;
- **Data Specificity**, riguardante il numero di parametri di una form con valori fissati.



3. TEMPLATE FINALI (2/2)

- Per non aumentare eccessivamente la Form Complexity si è deciso di non definire query che utilizzano tutta le clausole SQL' contemporaneamente. Per fare ciò il sottoinsieme SQL' è stato diviso al fine di avere quattro tipologie di query:
  - SELECT;
  - AGGR;
  - GROUP;
  - UNION/INTERSECT.

- Per non aumentare eccessivamente la Data Specificity si è deciso di non prendere in considerazione form che hanno parametri con valori fissati, per non far esplodere il numero di form risultanti.
  Inoltre possibili modifiche del database renderebbero invalide form con parametri fissati.

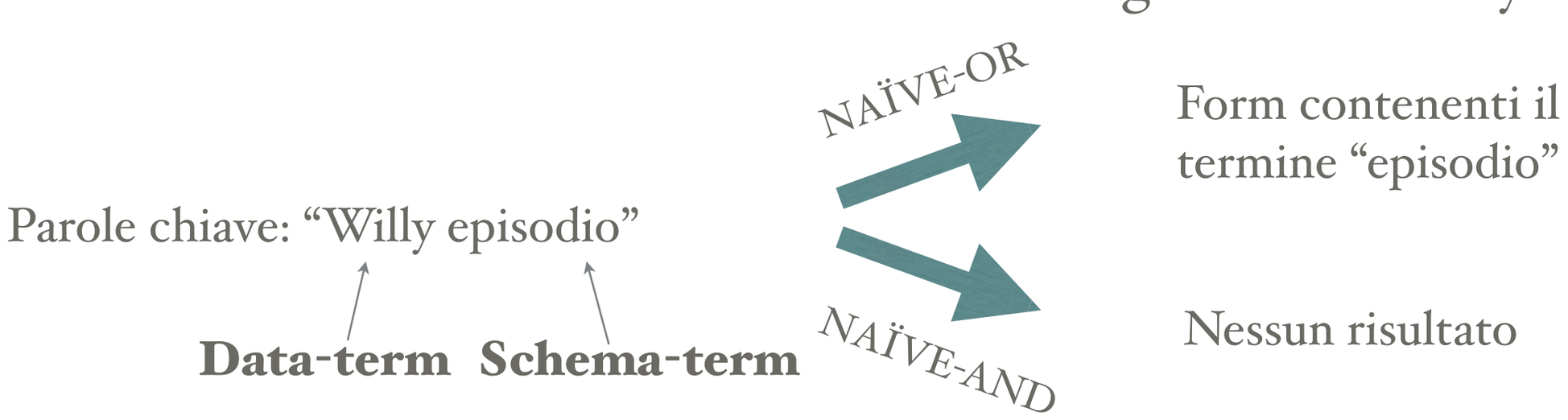
OUTLINE

- Come creare form che soddisfino il maggior numero possibile di query?
- Quanto sono efficaci le query nell'individuare le form più attinenti?
- Come mostrare i risultati per facilitare l'individuazione da parte dell'utente delle migliori form?
- I risultati ottenuti sono incoraggianti?

UN APPROCCIO INTUITIVO...

Si considera la form come un documento: l'utente immette le parole chiave, e vengono restituite le form che le contengono.

- ➡ NAÏVE-OR: restituisce le form che contengono almeno una keyword.
- ➡ NAÏVE-AND: restituisce le form che contengono tutte le keyword.



LIMITE: l'utente deve conoscere attributi e tabelle dello schema del database, perché ricerche con termini corrispondenti a valori interni alle tabelle non restituiscono risultati

UN APPROCCIO INTELLIGENTE...

IDEA DI BASE: interporre tra l'inserimento delle keyword e la ricerca della form, una fase di trasformazione delle parole chiave, che modifica la query con Schema-term rilevanti.

➡ DOUBLE-INDEX: basato sulla costruzione di due indici:

- **Data-Index**: data una keyword, restituisce una coppia <tuple-id,table> in cui tuple-id è la chiave primaria della tupla che contiene la keyword, e table è il nome della tabella;

Parola chiave: "Willy"

Personaggio

pid	nome	specie	sesso	nazione	abilità
1	willy	coyote	maschio	USA	predatore
2	beep-beep	road runner	maschio	USA	corridore
3	bugs bunny	coniglio	maschio	USA	furbizia
4	porky pig	maiale	maschio	USA	bontà
5	taddeo	uomo	maschio	USA	calma
6	yosemite sam	uomo	maschio	USA	pistolero

DataIndex(Willy) returns <1,Personaggio>

- **Form-Index**: dato un termine ritorna gli identificativi delle form che lo contengono.

ALGORITMO DOUBLE-INDEX OR

DI-OR trasforma la query aggiungendo tutti i termini dello schema corrispondenti a Data-term immessi dall'utente, e restituisce le form che ne contengono almeno uno.

Viene popolato il FormTerm con i termini della query e i nomi delle corrispondenti tabelle ottenuti dal Data-Index.

```
Input: query utente Q=[q1,q2,..,qn]
Output: un insieme di identificativi delle form F'
Algoritmo:
FormTerms = {}, F' = {}
//sostituzione di ogni data term con il nome della tabella
foreach qi ∈ Q
   if DataIndex(qi) returns <tuple-id,table>
      Aggiungi ogni table a FormTerms
   Aggiungi qi a FormTerms    // potrebbe essere un form term
//restituzione degli identificativi sulla base di FormTerms
FormIndex(FormTerms) => F'   // con semantica OR
return F'
```

Riscrittura query

Restituzione form

Interrogando il Form-Index con i termini presenti in FormTerm, vengono restituite le form che contengono **almeno uno** di questi termini

ALGORITMO DOUBLE-INDEX AND

DI-AND non solo applica le stesse modifiche di DI-OR, ma aggiunge alla query originale altre query che si ottengono sostituendo i Data-term con gli Schema-term corrispondenti. Per ognuna di queste query utilizza la semantica AND e restituisce l'unione dei risultati.

```
Input: query utente Q=[q1,q2,...,qn]
Output: un insieme di identificativi delle form F'
Algoritmo:
FormTerms = {}, F' = {}
//sostituzione di ogni data term con il nome della tabella
foreach qi ∈ Q
   Sqi = {}                          // Bucket per qi
   if DataIndex(qi) returns <tuple-id,table>
      foreach table
         if table ∉ FormTerms
         Aggiungi table a Sqi e a FormTerms
   if qi ∉ FormTerms
      Aggiungi qi a Sqi e a FormTerms
//generazione di tutte le query univoche
//aventi un termine preso da ogni Sqi
SQ' = EnumQueries(per ogni Sqi)
//restituzione degli identificativi sulla base di Sqi
foreach Q' ∈ SQ'
   FormIndex(Q') => F'    // con semantica AND
return F'
```

Per ogni termine della query, viene creato un bucket, che conterrà il termine e i nomi delle tabelle associate ad esso. Inoltre, vengono generate query univoche che contengono un termine per ogni bucket.

Riscrittura query

Per ogni query generata viene interrogato il Form-Index che restituisce le form che contengono **tutti** i termini della query.

Restituzione form

DI-OR E DI-AND ALL'OPERA

Parole chiave: "Willy episodio"

DI-OR

Data-Index(Willy) returns <1, Personaggio>
Data-Index(episodio) returns nothing

Nuova query = "Willy personaggio episodio"

Restituzione form contenenti Willy OR Personaggio OR Episodio.

DI-AND

Data-Index(Willy) returns <1, Personaggio>
Data-Index(episodio) returns nothing

Bucket(Willy) = {Willy, Personaggio}
Bucket(episodio) = {episodio}

Nuove queries = "Willy episodio", "personaggio episodio"

Restituzione form contenenti Willy AND episodio o Personaggio AND episodio.

LIMITE DOUBLE-INDEX

LIMITE: quando la ricerca coinvolge una tabella t_I referenziata da tabelle t_i, DI restituisce oltre alle form corrispondenti a t_I anche quelle corrispondenti alle t_i. Esse potrebbero non contenere i Data-term della query e non restituirebbero alcun risultato; per questo vengono chiamate **Dead-form**.

Parole chiave: “Porky Pig”

DataIndex(Porky Pig) returns <4,Personaggio>

↓

Restituzione form relativa alla tabella personaggio e alle tabelle che la referenziano: Episodio_Personaggio e Nemici.

↓

L'identificativo di Porky Pig non è presente nella tabella “Nemici”, quindi tutte le form inerenti a questa risultano essere Dead-form

Personaggio

pid	nome	specie	sesso	nazione	abilità
1	willy	coyote	maschio	USA	predatore
2	beep-beep	road runner	maschio	USA	corridore
3	bugs bunny	coniglio	maschio	USA	furbizia
4	porky pig	maiale	maschio	USA	bontà
5	Taddeo	uomo	maschio	USA	calma
6	Yosemite Sam	uomo	maschio	USA	pistolero

Nemici

nid	pid1	pid2
1	1	2
2	3	5
3	3	6

ALGORITMO DOUBLE-INDEX JOIN

DIJ applica una modifica a DI-AND in modo che le Dead-form vengano scartate.

```
Input: query utente Q=[q1,q2,...,qn]
Output: un insieme di identificativi delle form F'
Algoritmo:
FormTerms = {}, F' = {}, X = {}
//sostituzione di ogni data term con il nome della tabella
foreach qi ∈ Q
   Sqi = {}                          // Bucket per qi
   if DataIndex(qi) returns <tuple-id,table>
      foreach table T
         sia I l'insieme di tuple-ids di T
         if T ∉ FormTerms
         Aggiungi T a Sqi e a FormTerms
         //nuovo step di join
         SchemaGraph(T) returns refTables
         foreach refTable
            if DataIndex(refTable:tid) is NULL per ogni tid ∈ I
               FormIndex(T AND refTable) => X
   if qi ∉ FormTerms
      Aggiungi qi a Sqi e a FormTerms
//generazione di tutte le query univoche
//aventi un termine preso da ogni Sqi
SQ' = EnumQueries(per ogni Sqi)
//restituzione degli identificativi sulla base di Sqi
foreach Q' ∈ SQ'
FormIndex(Q') => F'    // con semantica AND
return F' - X            // eliminazione Dead-form
```

Sfruttando il Data-Index per ogni tabella T abbiamo un insieme I di tuple-ids; con l'operazione SchemaGraph(T)) si memorizzano tutte le tabelle che referenziano T. Per ogni refTable si controlla che non contenga le chiavi primarie in I; se ciò accade tutte le form che contengono sia T che refTable vengono dichiarate Dead-form e per questo in seguito scartate.

Individuazione Dead-form

OUTLINE

- Come creare form che soddisfino il maggior numero possibile di query?
- Quanto sono efficaci le query nell'individuare le form più attinenti?
- Come mostrare i risultati per facilitare l'individuazione da parte dell'utente delle migliori form?
- I risultati ottenuti sono incoraggianti?

RANKING DELLE FORM

- Per mostrare le form risultanti in modo adeguato si è pensato di trattarle come documenti e applicare una funzione di ranking utilizzando un indice Lucene.
- **Lucene** è una API gratuita ed open source sviluppata da Apache Software Foundation, concepita per realizzare applicazioni che necessitano di funzionalità di indicizzazione.
- L'indice Lucene permette di assegnare ad ogni coppia <query, documento> un punteggio **TF/IDF** normalizzato.
- TF/IDF è una misura statistica usata per valutare quanto è importante un termine in un documento:
  - TF (Term Frequency) normalizzato è il rapporto tra il numero di occorrenze del termine *t* nel documento *j* e la frequenza massima di un termine *l* in quel documento: $$tf_{t,j} = \frac{freq_{t,j}}{\max_l\{freq_{l,j}\}}$$
  - IDF (Inverse Document Frequency) è negativamente correlato con il numero di documenti nel quale il termine *i* appare, ed è calcolato tramite il logaritmo del rapporto tra il numero totale di documenti *N* e il numero di documenti N_t in cui il termine è presente: $$idf_t = \log\left(\frac{N}{N_t}\right)$$
- La funzione di score di Lucene per una query Q e un documento D è:

$$score(Q,D) = coord(Q,D) * queryNorm(Q) * \sum_{t\ in\ Q} (tf(t\ in\ D) * idf(t)^2 * t.getBoost(\ ) * norm(t,D))$$

punteggio basato sul numero di termini in Q nel documento D

fattore di normalizzazione

search-time boost di t, settato a 1

index-time boost di t in D, settato a 1

RAGGRUPPAMENTO FORM (1/2)

PROBLEMA: la specificità delle form incrementa il numero di **form "sorelle"**, cioè basate sullo stesso template scheletro. La presenza di troppe form sorelle va a svantaggio dell'usabilità, poiché l'utente non sarà in grado di discriminare facilmente i risultati.

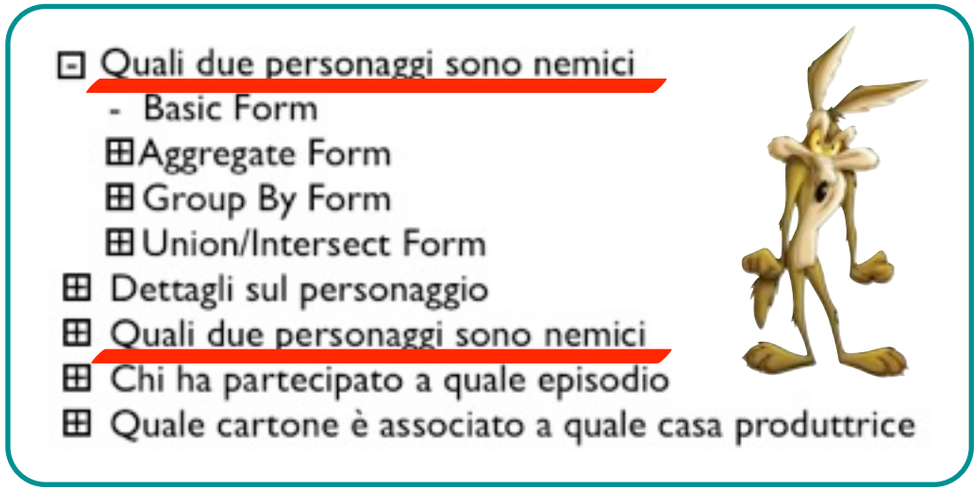
INTUIZIONE: collassare le form sorelle in un unico gruppo, in modo che l'utente possa decidere quale gruppi esplorare nel dettagli in base alle informazioni che sta cercando.

- Metodo basato sui punteggi
- Metodo basato sulle tabelle

RAGGRUPPAMENTO FORM (2/2)

✳ METODO BASATO SUL RANKING

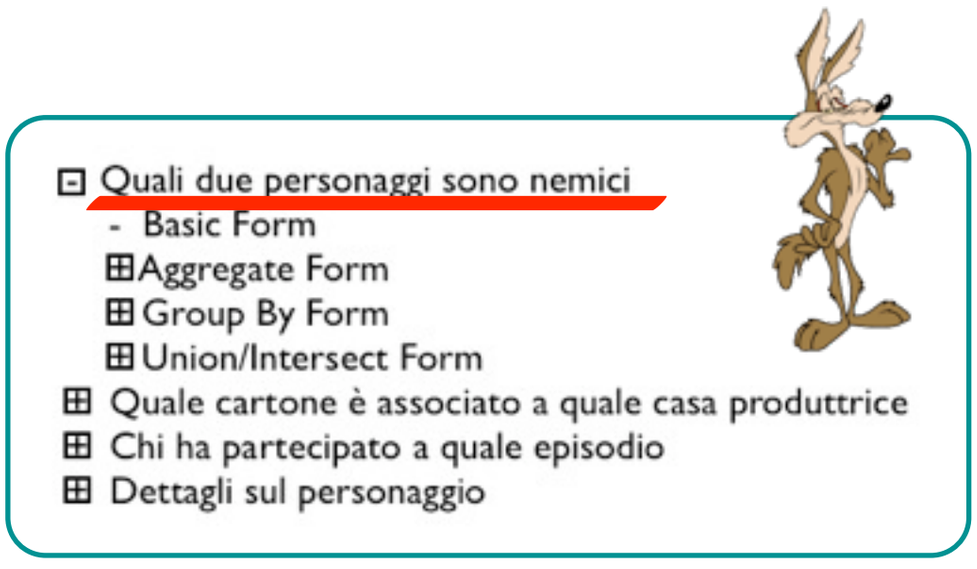
- gruppi di primo livello: raggruppamento delle form “sorelle” consecutive con lo stesso punteggio.
- gruppi di secondo livello: raggruppamento delle form in base alle 4 classi di query (SELECT, AGGR, GRUOP, UNION/INTERSECT) all’interno di ogni gruppo di primo livello.



LIMITE: sister form non consucutive o con diverso punteggio apparterranno a gruppi di primo livello differenti.

✳ METODO BASATO SULLE TABELLE

- gruppi di primo livello: raggruppamento delle form in base alle tabelle a cui si riferiscono.
- ordinamento dei gruppi in base alla somma dei punteggi delle form al loro interno.
- gruppi di secondo livello: raggruppamento delle form in base alle 4 classi di query (SELECT, AGGR, GRUOP, UNION/INTERSECT) all’interno di ogni gruppo di primo livello.



OUTLINE

- Come creare form che soddisfino il maggior numero possibile di query?
- Quanto sono efficaci le query nell'individuare le form più attinenti?
- Come mostrare i risultati per facilitare l'individuazione da parte dell'utente delle migliori form?
- I risultati ottenuti sono incoraggianti?

RISULTATI SPERIMENTALI (1/3)

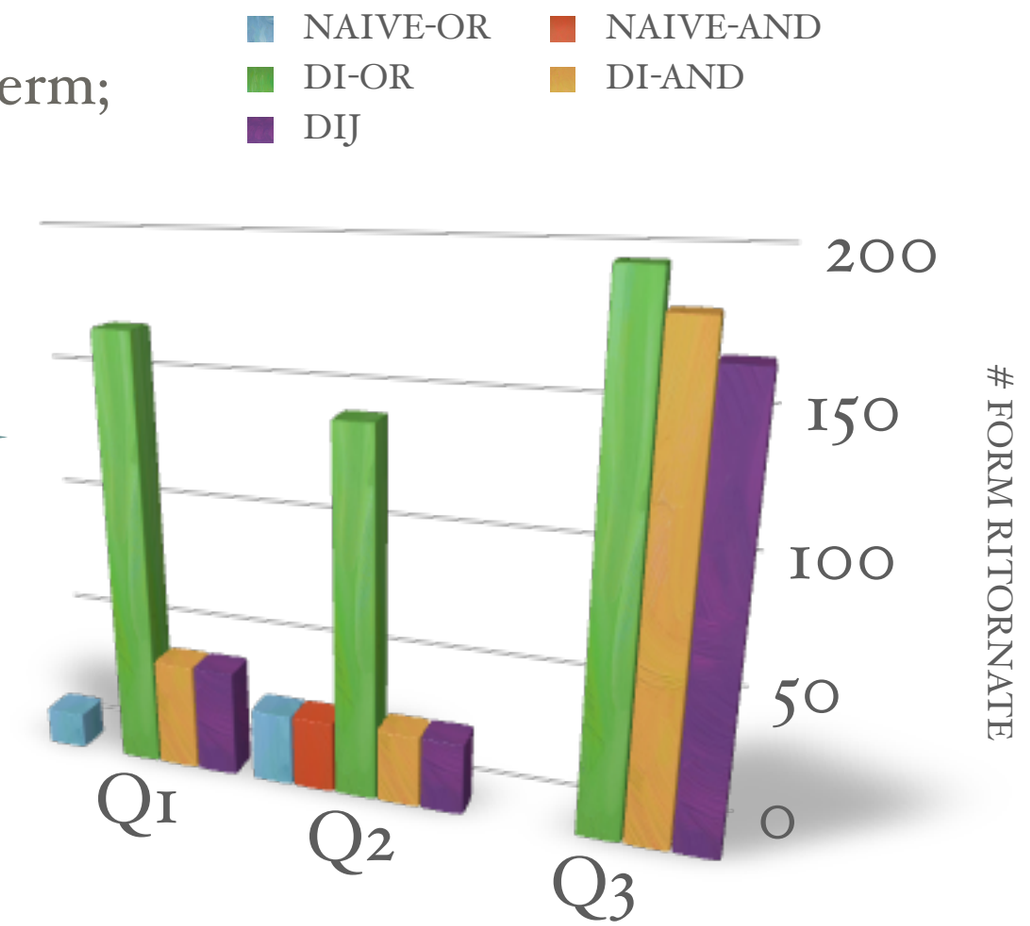
- Impostazioni di sistema: Red Hat Enterprise Linux 5 workstation con 2.33 GHz CPU e 3GB RAM
- Benchmark: DBLife formato da 5 tabelle di entità e 9 tabelle delle relazioni, con un totale di 801189 tuple.
- Sono stati creati 14 template scheletri, uno per ogni tabella; per ognuno sono stati creati un SELECT template, 5 AGGR template, 6 GROUP template e 2 UNION/ INTERSECT, per un totale di 196 template.

Consideriamo tre query:

- Q1 contenente un Data-term e uno Schema-term;
- Q2 contenente due Schema-term
- Q3 contenente due Data-term

Il grafo mostra il numero di form restituite da ogni algoritmo per ogni query:

- NAIVE-AND non ritorna form per Q1 e Q3, poiché sono presenti Data-term;
- DI-OR non è efficiente poiché ritorna per Q3 tutte le 196 form;
- DI-AND e DIJ hanno comportamenti simili tranne per Q3 dove DIJ eliminando le Dead-form riduce il numero delle form da 182 a 168.



RISULTATI SPERIMENTALI (2/3)

La tabella mostra la posizione in cui viene visualizzata la form migliore, senza utilizzare i metodi di raggruppamento:

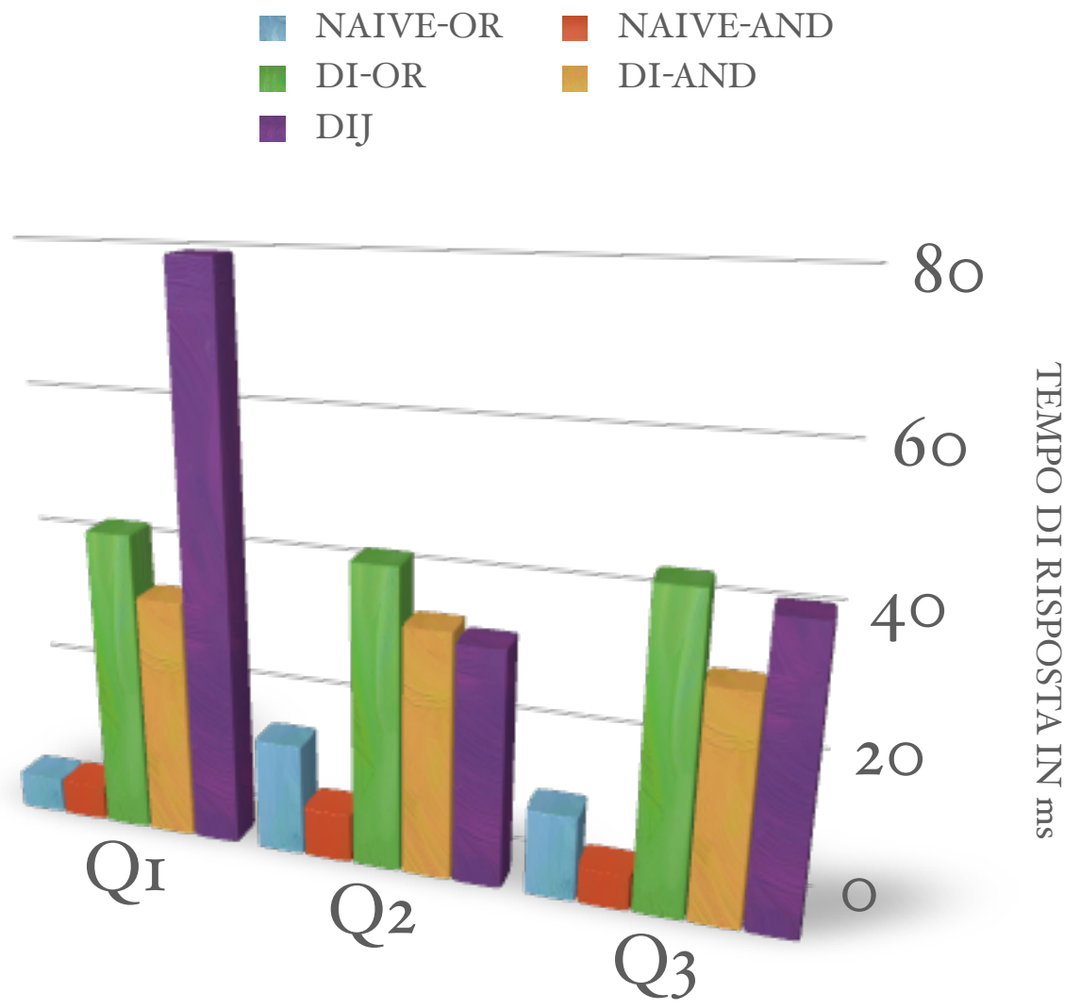
- dove non vengono restituiti i risultati l'ordinamento non è applicabile;
- DIJ è più efficace per Q3 poiché eliminando le Dead-form permette alla Best-form di scalare il Rank da 27 a 12.

	NAIVE-OR	NAIVE-AND	DI-OR	DI-AND	DIJ
Q1	1	N/A	29	1	1
Q2	1	1	29	1	1
Q3	N/A	N/A	51	27	12

Il grafico mostra il tempo di risposta dei diversi algoritmi in millisecondi:

- i metodi NAÏVE risultano più veloci poiché non eseguono la riscrittura della query;
- i tempi di risposta dei metodi DI sono significativamente maggiori, tuttavia poiché la differenza è di qualche decina di ms essa non è percepibile dall'utente umano.

Vince DIJ!!



RISULTATI SPERIMENTALI (3/3)

	SENZA RAGGRUPPAMENTO				CON RAGGRUPPAMENTO (METODO PUNTEGGI)			
	H	M	L	#FORM	H	M	L	#GRUPPI
Q1	1	1	1	44	1	1	1	3,14
Q2	1	15	15	28	1	2	2	2
Q3	1	12	12	56	1	1	6	4

La tabella prende in considerazione l'immissione delle keyword da parte di sette utenti e mostra la posizione più alta (H), mediana (M) e più bassa (L) in cui la form o il gruppo corretto vengono mostrati, usando l'algoritmo DIJ. Inoltre viene riportata il numero medio di form e gruppi restituiti:

- Senza raggruppamento la form corretta potrebbe avere differenti posizioni a secondo delle diverse keyword immesse dagli utenti;
- Con il raggruppamento, invece, la form corretta appartiene sempre ai primi 6 gruppi, facilitandone l'individuazione da parte dell'utente.

CONCLUSIONI

- L'intento di questo lavoro consiste nel creare un ponte tra l'accesso ad un database e la ricerca di informazioni tramite parole chiave;

- Importante è valutare il tradeoff tra la ricerca con parole chiave che restituisce direttamente risultati, talvolta imprecisi, e l'approccio presentato che, introducendo un livello di intermediazione, mostra una lista ordinata di possibili interrogazioni;

- Questo obiettivo è stato perseguito tramite l'utilizzo di form che cercano nascondere i dettagli di un linguaggio di interrogazione ad un utente non esperto.

SVILUPPI FUTURI

- Generazione di tecniche semi-automatiche per avere descrizioni delle form attinenti al contenuto delle tabelle cui si riferiscono;
- Creazione dinamica di template scheletri, che permettano al sistema di scalare database di grandi dimensioni;
- Realizzazione di tecniche che permettano di apprendere dal feedback che l'utente fornisce riguardo alle form più rilevanti.

That's all Folks!

Votate Gruppo 15!!