1a

02/10/2012

MODELLI AVANZATI E ARCHITETTURE DI BASI DI DATI

(PANORAMICA DEL CORSO)

* Sono necessari meccanismi di indicizzazione, controllo della concorrenza e recovery

MODELLI AVANZATI:

Rivoluzionario → (OODB) BASI DI DATI

Evolutivo → BASI DI DATI

- STRUTTURATO: OBJECT-ORIENTED / OBJECT-RELATIONAL
  - Gli oggetti sono PERSISTENTI e possono essere recuperati e condivisi da altri programmi *
  - ↳ Aggiunge al mod. Rel. costrutti object-oriented
- NON STRUTTURATO: IR noti come SEARCH ENGINES (Google) = Motore di ricerca

IR = Information Retrieval = Recupero delle inform. ↳ Indirizzano mole di dati presente non strutturata

LIBRI DI TESTO:

- ELMASRI, NAVATHE Sistemi di basi dati. Complementi (2005)
- BAEZA-YATES, RIBEIRO-NETO Modern Information Retrieval
- SACCO, TZITZIKAS (ED.) Dynamic Taxonomies Faceted Search Springer

Modelli di IR

Primo sistema (uno dei primi) è SMART-SALTON

modello vettoriale (ALGEBRICO): Doc. e query sono rappresentati da vettori in uno spazio t-dimensionale
↳ Funzione di somiglianza: coseno dell'angolo → Se sono uguali fa 1, diversi 0, ciò che si sta in mezzo: somiglianza

modello booleano (SET THEORETIC): I documenti e le query sono rappr. da un insieme di termini di indicizzazione (?)

modello deviazione dalla randomness (PROBABILISTICO)
↳ Ignora un numero limitato di termini (tipo articolo il)
Problema: quali termini sono rilevanti?

Eliminando i termini inutili otteniamo strutture di indicizzazione più compatte

Tutto ciò che è rilevante viene considerato

Solo ciò che è rilevante viene considerato

VALUTAZIONE dell'output di una SEARCH ENGINES
* 2 tipi di valutazione:

Recal (Esaurienza del risultato): Rapporto tra # dei doc. rilevanti recuperati e il tot dei doc. della raccolta

Precision (Ampiezza di pertinenza nel risultato): Rapporto tra il # dei doc. rilevanti recuperati e il tot dei doc rilevanti della raccolta

AMBIENTE WEB

- CRAWERS/ROBOT: Programmi che visitano tutti i siti, verificano cosa è cambiato dall'ultima visita e aggiornano la banca dati
- MULTILINGUA
- PAGE RANKING

↳ Algoritmo usato da google. Non si basa solo sulla frequenza del term nel documento ma anche sul # di riferimenti del documento ("Autorevolezza della pagina")

Dal punto di vista dell'interazione UOMO/MACCHINA sono state aggiunte altre cose come: AUTO COMPLETE e SUGGERIMENTO DI TERMINI DI RICERCA

- DATI STRUTTURATI: Dati parzialmente strutturati (Sistema XML)

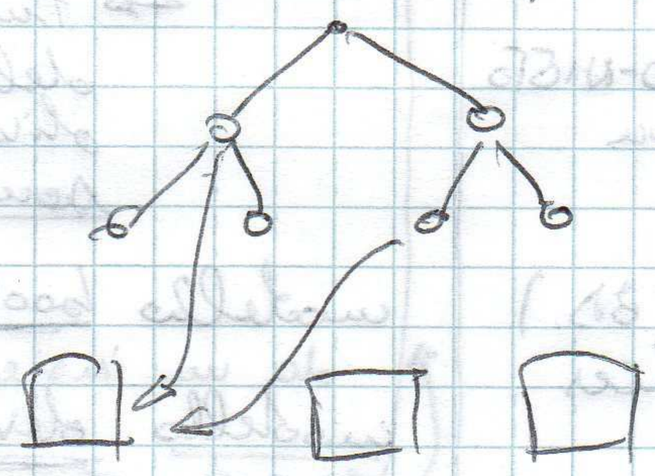
Vedremo bene DYNAMIC TAXONOMIES (Tassonomia Dinamica)

Detta anche FACETED SEARCH

Usa le informazioni di classificazione per affinare l'albero

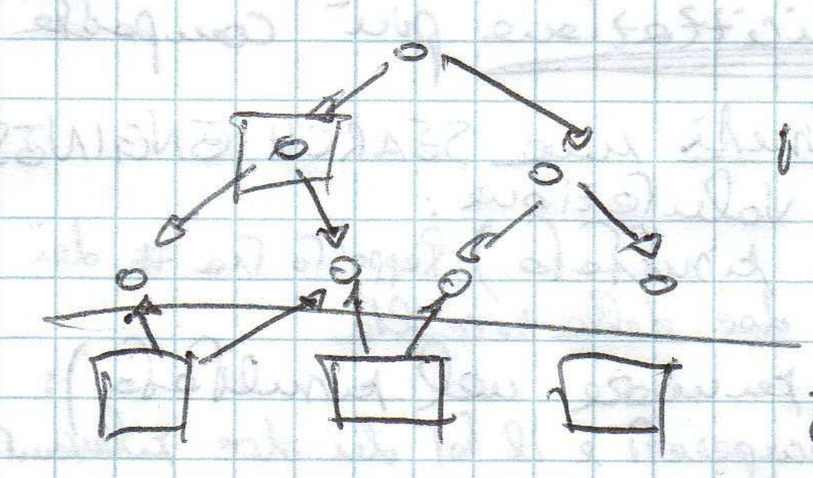
Sistema di gestione della conoscenza (Vicini ai motori di ricerca): permettono un accesso guidato all'informazione. Basati su una classificazione dei documenti sulla base di una tassonomia (Struttura che va da un concetto generale a quello più specifico es. classi e sottoclassi).

I documenti sono classificati sotto più parti



La classificazione può essere fatta per tempo, luogo e argomento

Es. Ho 3 documenti



Il sistema dinamico forma indirettamente per prendere i concetti usati per classificare ciò che ho recuperato (frecce in su)

Il sistema adatta alla tassonomia il focus di interesse eliminando dei rami inutili alla ricerca.

Si può raffinare ulteriormente iterando; VANTAGGIO rispetto alla tassonomia non dinamica.

Si trova su tutti i siti e-commerce.
Questo tipo di sistema permette una ricerca esplorativa in cui l'utente non sa esattamente cosa sta cercando e quindi viene guidato nella ricerca. Tale meccanismo viene usato implicitamente nella diagnosi medica.

2a

Introduzione ai

03/10/2012

MODELLI AVANZATI: Modelli di dati tradizionali

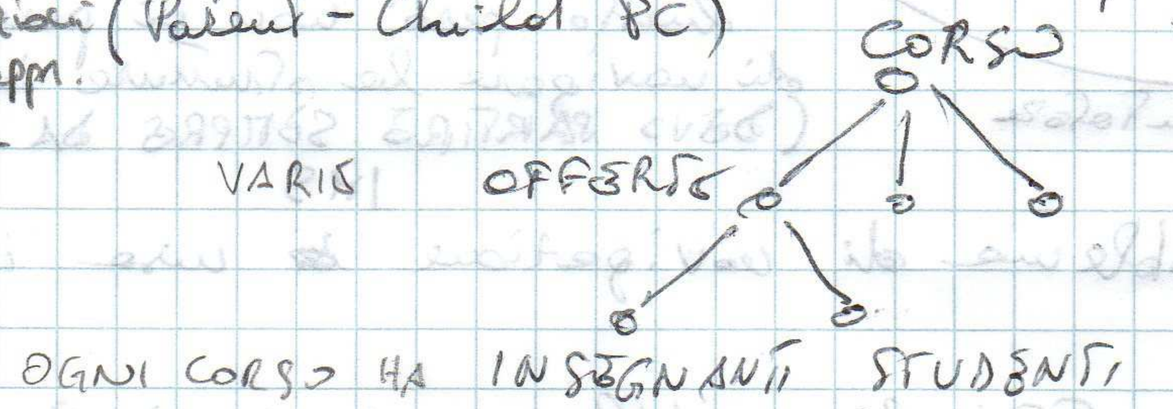
Modelli Proposti:

GERARCHICO → (IBM - IMS) Sistema. ANNI '60. Si utilizzavano i nastri magnetici (Accesso sequenziale)

↓

I dati sono rappr. come RECORD e le associazioni tra i dati vengono rappr. con puntatori in una struttura ad albero

A partire dalla radice si possono avere vari figli (Parent - Child PC)



la gerarchia può essere profonda quanto si vuole

VANTAGGI: linearizzazione dell'albero (IMPORTANTE x NASTRO)
↳ lavora in maniera efficiente con i nastri

SVANTAGGI: le relazioni sono solo 1 a N.

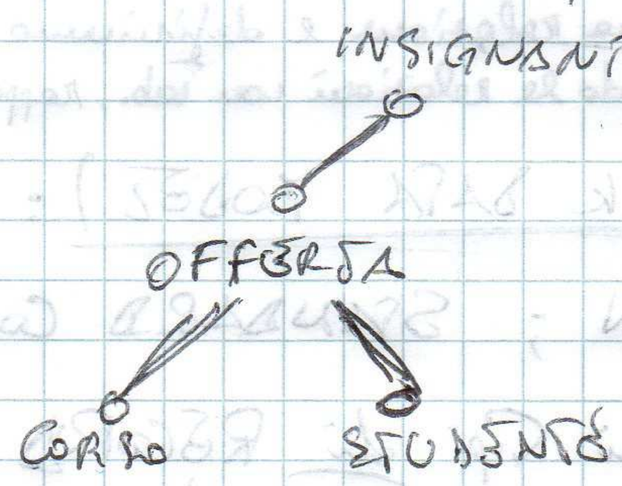
C'è una notevole asimmetria

L'ASIMMETRIA è il fatto che ho un ordine di accesso predefinito. Alcune query risolve bene, altre male. (Es. Ricerca degli insegnanti)
Per effettuare delle ricerche devo attraversare tutto l'albero

ANOMALIE: Uno studente rimane nel limbo finché non si iscrive ad un corso.

Risoluzione ASIMMETRIE (Evoluzione sistema gerarchico)

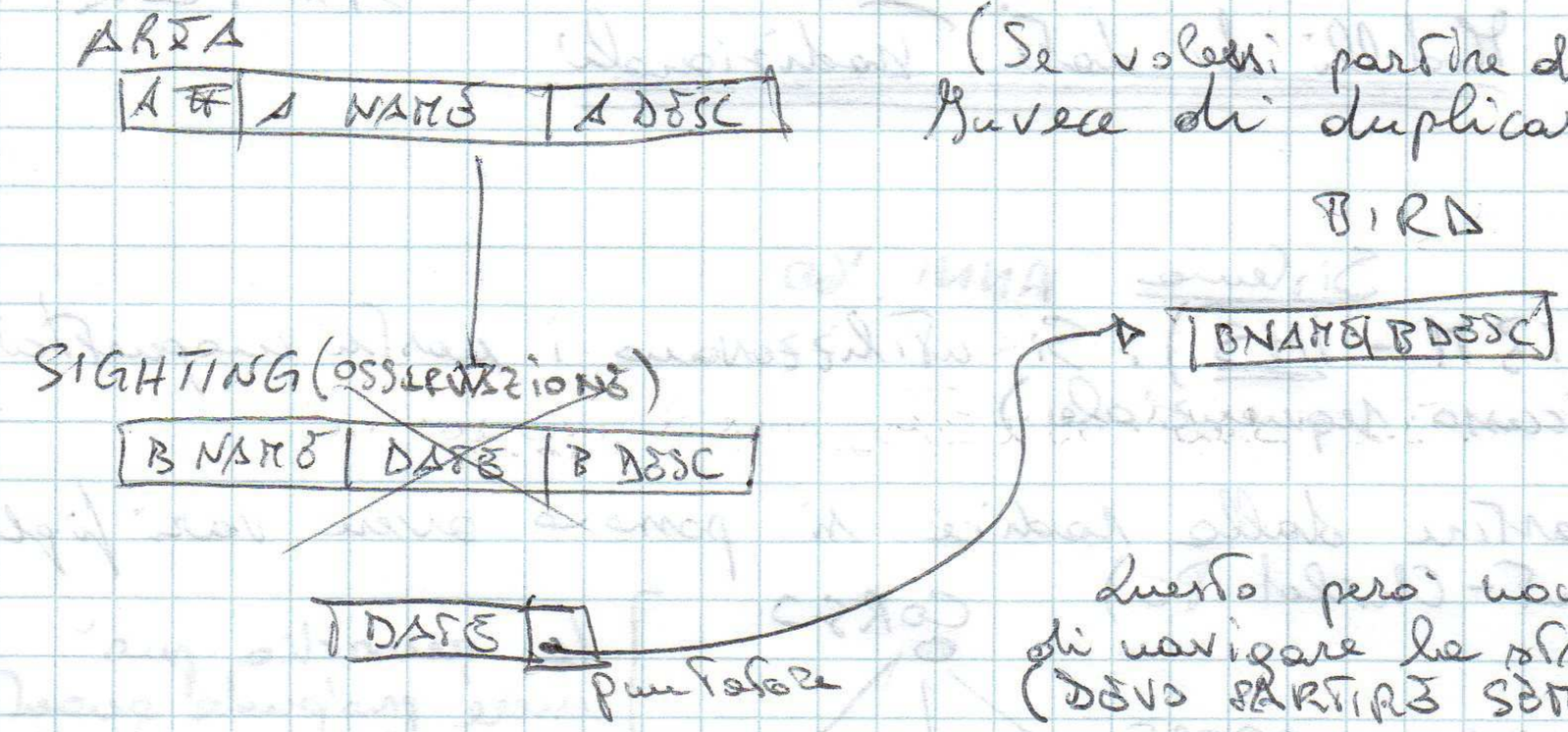
DUPLICARE I DATI → Si costruisce una nuova gerarchia con diversa radice. Dunque si hanno più gerarchie con radici diverse ottenendo però una consistente RIDONDANZA DEI DATI.



STORAGE ESTREMAMENTE COSTOSO

Per risolvere questo problema IMS introduce dei meccanismi di puntamento.
Invece di duplicare i dati vengono create più gerarchie collegate tra di loro tramite puntatori.

Esempio (Database di osservazioni ornitologiche)



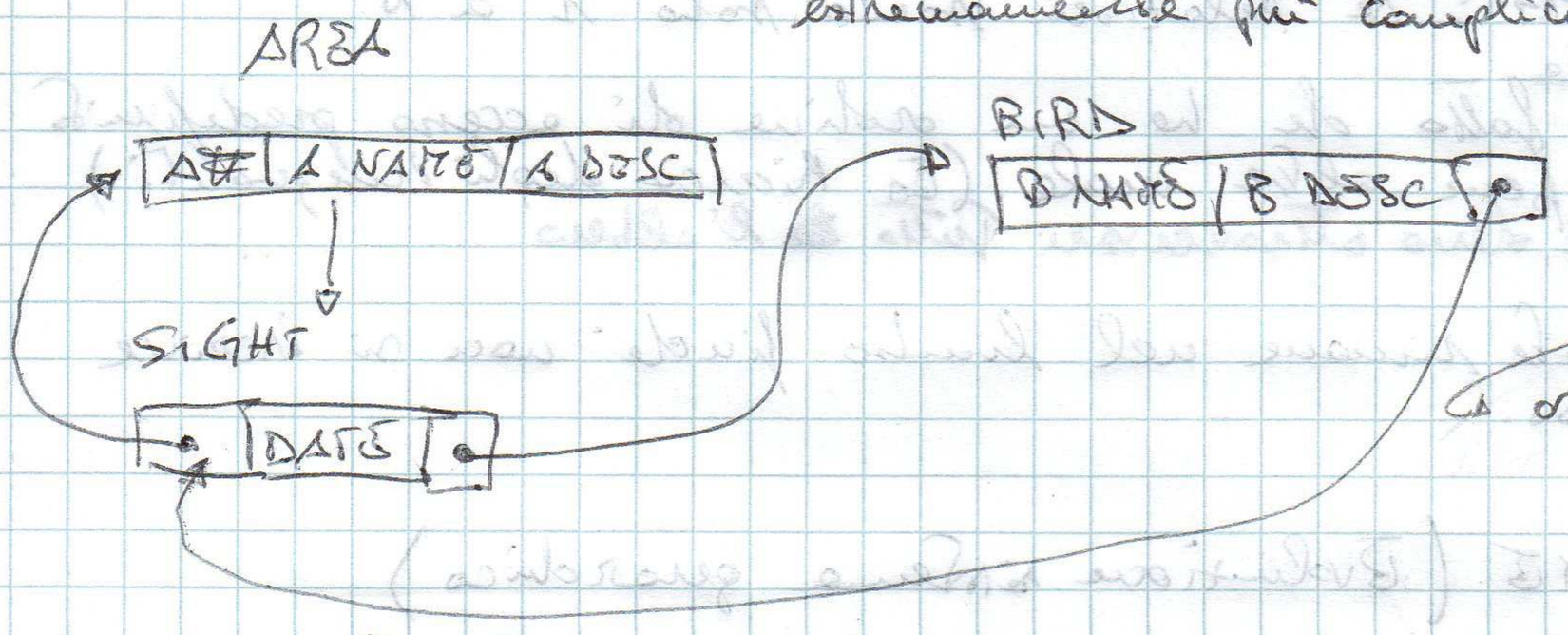
(Se volessi partire dall'osservazione e non dall'area) Invece di duplicare con una radice diversa, tolgo i dati BNAME e BDESC dall'osservazione lascio la data e aggiungo un puntatore che punta alla descrizione dell'uccello (osservazione).

Questo però non ci permette, comunque, di navigare le strutture in modo più libero (DOVO PARTIRE SEMPRE DA AREA)

IMS

Per risolvere il problema di navigazione ~~si~~ uso il meccanismo:

VIRTUAL PAIRING : Stabilisce delle associazioni logiche bidirezionali. Rende tutto navigabile in modo arbitrario (posso partire da dove voglio) ma diventa tutto estremamente più complicato.



Mediante i puntatori si evita delle gerarchie di accesso senza duplicare i dati.

↳ 0 cammini di accesso

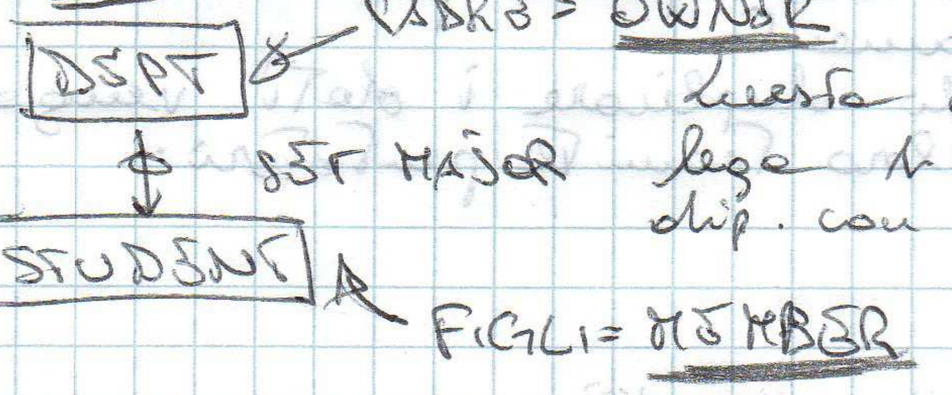
Questo tipo di puntatori non si possono utilizzare con i nastri ma con i dischi (che permettono l'accesso diretto). Vedendo le cose come Relazioni si nota che partiamo definendo una relazione e definiamo poi esplicitamente un inverso. Nel relazionale invece, rappresentando le relazioni con tab., rappresentiamo immediatamente l'associazione e ogni suo inverso.

MODELLO RETICOLARE (NETWORK DATA MODEL):
(ANNI '70)

DBTG proposto da BACHMAN ; STANDARD CODASYL (Evoluzione del Modello Reticolare)

Si basa su puntatori. Dati dei tipi di RECORD, si cerca di trovare una rappresentazione che permetta di realizzare associazioni di qualunque tipo. Anche M-N

Es.



Questa associazione lega 1-N il dip. con lo studente

Si parte sempre da 1-N

[I dati vengono rappresentati come RECORD e le associazioni tra i dati sono rappr. con puntatori in una struttura a grafo.]

SET = ASSOCIAZIONE TRA OWNER E MEMBER

Es.

3a ⟶ Data la struttura di memorizzazione dell'es. di prima la Rappresentazione fisica è l'enumerazione dei vari dipartimenti

CS → HUMAN → BIOLOGY

S_1

S_2

S_n

Gli stud. sono legati tramite puntatori tra di loro e l'ultimo studente è ricollegato al padre

Le strutture a puntatori vengono attraversate per risolvere le interrogazioni (Avviene la stessa cosa nei sistemi Hypertextuali

In questa struttura c'è una notevole connessione con l'Hypertesto

Se si vogliono enumerare tutti gli stud., partiamo dal 1° dip, scantiamo e li enumeriamo; passiamo al 2° etc etc

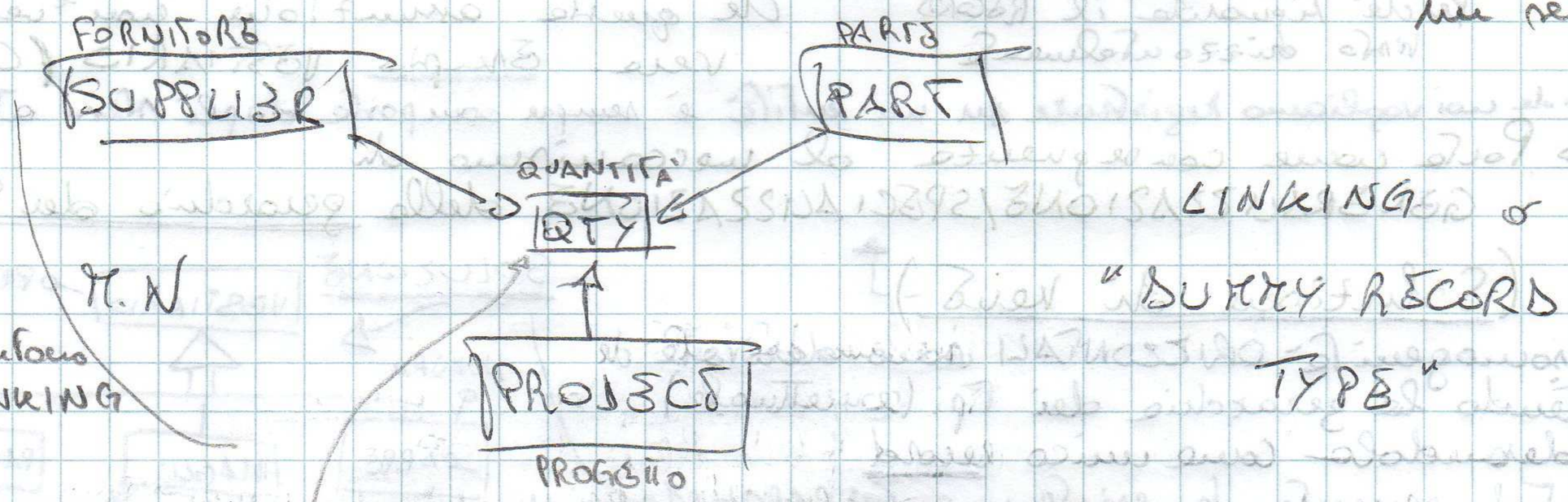
In questo modello si hanno i MULTIMEMBER SETS → oggetti diversi che partecipano alla stessa associazione. I membri di un'associazione possono essere di tipi diversi

Si hanno anche i SET RECURSIVI* e i SET SPECIALI il cui OWNER è il sistema vero e proprio. ← Vengono tipicamente usati per enumerare tutti i dati

In questo modello l'1 a 1 si implementa con un singolo. (non si permette ad un livello di avere più di un MEMBER)

L'M-N non è banale.

Supponiamo di avere questa struttura che non siamo in grado di rappr. con un ref.



Si utilizza un record Type fittizio che funziona come MEMBER di più OWNER (LINKING "DUMMY RECORD TYPE") In questo modo rappresentiamo relazioni M-N generiche

SVANTAGGI
- Questo tipo di modello è completamente procedurale. La navigazione nel database è procedurale (PROBLEMA!) - Il programmatore "segue" i puntatori
- Non esiste separazione tra aspetto ~~macro~~ logico e fisico (ALTRO PROBLEMA!) Il programma ha visibilità logica e fisica di tutto. Es. Se PROJECT è rappr. con un B-TREE e voglio passare all'hashing devo cambiare programma

Col modello RELAZIONALE tutta la complessità di questi sistemi viene eliminata. (CODD)

Col RELAZIONALE vengono separate la parte LOGICA da quella FISICA → SCHEMA LOGICO e SCHEMA FISICO

Codd definisce un modo matematico rigoroso (Alg. Rel.), non procedurale, per definire il risultato di un'interrogazione

* Es. IMPIEGATO e SUPERVISORE → Il SUPERVISORE è un IMPIEGATO	Es. Un set che enumera tutti i DIPARTIMENTI

3b MODELLO RELAZIONALE: CODD 1970 (Entra nel mercato massicciamente nella seconda metà degli '80)
↳ Proposta '70

- Semplice ma COMPLETO
↳ Dal punto di vista CONCETTUALE e della MANIPOLAZIONE

Indipendenza fra SCHEMA LOGICO e FISICO

VANTAGGI: Non Proceduralità - Separazione LOGICO-FISICA - SET-AT-A-TIME (Recuperiamo insiemi di dati e non un dato per volta)

DIFFICOLTÀ: - Tutto era sotto il modello RETICOLARE → Migrazione dei dati.

I primi sistemi relazionali sono: - SYSTEM-R (IBM) e INGRES
↳ di cui ORACLE è una copia esatta ↳ Università di California

- I primi sistemi relazionali erano "DEI CHIODI" soprattutto con i JOIN.

ARTICOLO - W. KENT (Lettura non obbligatoria): LIMITATIONS OF RECORD-BASED INFORMATION MODELS - TODS 4.1, MARCH 1979

Critica tutti i modelli basati sui RECORD quindi anche tutti i modelli presentati precedentemente. (GERARCHICO, RETICOLARE e RELAZIONALE)

Definizione di Kent di RECORD:

"Il record è una sequenza fissa di campi conformi che sono definiti da una descrizione statica" (FATTA A PRIORI)

Le obiezioni che fa a una struttura di questo tipo sono 2:

1. ORIZZONTALE (ANOMALIA) → Omogeneità dei fatti rilevanti* Dice che questa assunzione non è necessariamente vera. Esempio VESTIARIO (CLOTHING)
Perché riguarda il RECORD visto orizzontalmente

Dalla def. di RECORD si ha:

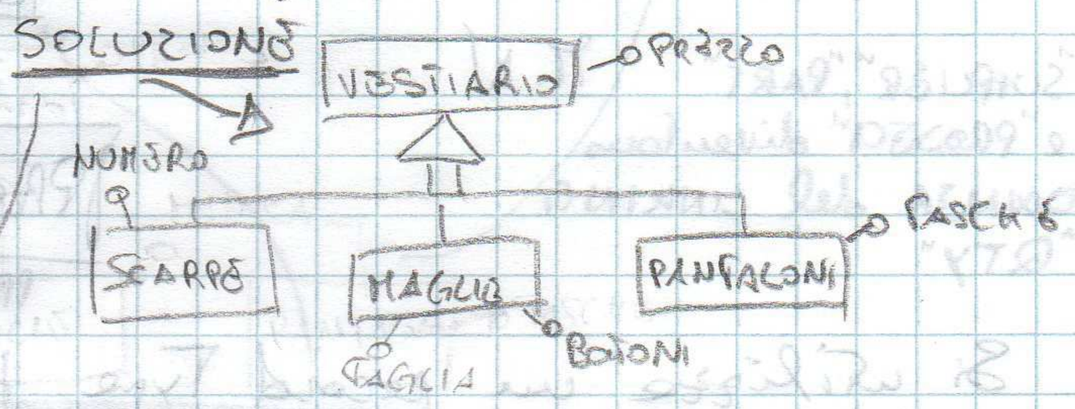
*Ossia Quello che noi vogliamo registrare su un'entità è sempre composto dagli stessi attributi

→ Porta come conseguenza al meccanismo di GENERALIZZAZIONE/SPECIALIZZAZIONE della gerarchia dei tipi

Soluzione di Kent

Le disomogeneità ORIZZONTALI derivano dal fatto che appiattiamo la gerarchia dei tipi (concettuale) considerandola come unico record
↳ Kent fa presente che esiste una GERARCHIA DI TIPI

SOLUZIONE

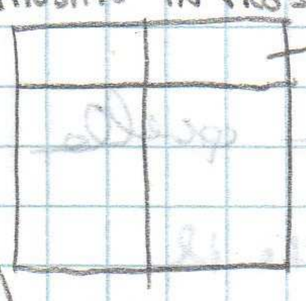


2. ANOMALIA VERTICALE → In un tipo di RECORD un campo contiene sempre lo stesso tipo di valori. (NON VA BENE).

Ci sono situazioni pratiche in cui oggetti diversi partecipano ad una stessa associazione.

Es. Un azionista può essere → Persona fisica / → Persona Giuridica
- Persona fisica → COD FISCALE
- Persona Giuridica → PARTITA IVA
Possono essere identificate in maniera diversa

OGG. IN PRESTITO "OGG" CHE PRENDE IN PRESTITO



→ Può essere Stud, altro dipartim. (Possono avere ID diversi)

OGGETTO IN PRESTITO
Può essere nel caso di un dipart. → DIPARTIMENTO / → IMPIEGATO
Un oggetto può essere preso in prestito da un dip. o un imp. che vengono identificati diversamente
AULA, PROIETTORE, LIBRO, etc. (Possono avere ID diversi)

ESR → CATEGORIE

IL PROBLEMA sta nel fatto che in molti casi vogliamo usare un unico campo per rappresentare tipi diversi

1a

Le conseguenze delle critiche di Kent sono:

- Per ciò che riguarda le disomogeneità ORIZZONTALI:

1) Reticolo semantico (GERARCHIA DI TIPI già accennata)

↳ COME CONCETTO DI MODELLIZZAZIONE CHE DEVE ESSERE SUPPORTATO DAL MODELLO

a) Ereditarietà multipla dei dati con i problemi noti di ambiguità

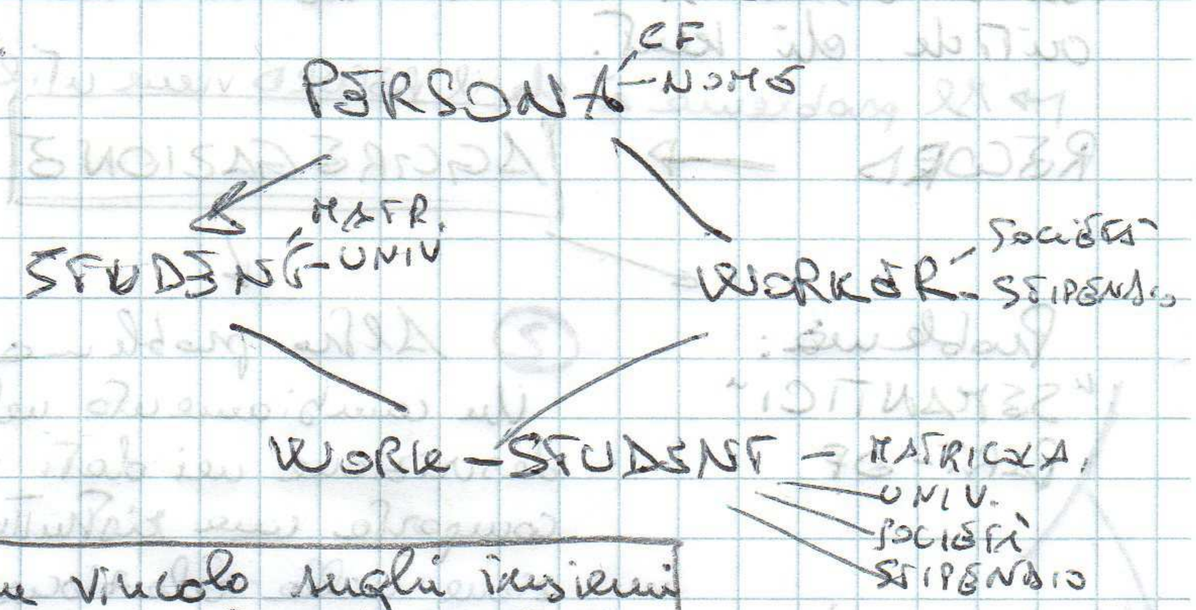
b) Proprietà di inclusione nella istanza (SUBSUMPTION)

A < B (B SUBSUMES A)

implica $ist(A) \leq ist(B)$

↳ Nell'esempio precedente: le istanze di STUDENTE sono un sottoinsieme delle istanze di PERSONA.

Es.



È un vincolo sugli insiemi che rappresentano le istanze in questo reticolo di dati

Applicando questa struttura ottengo:

PERSONA: CF, NOME, [UNIV, MAT], [SOCIETÀ, STIP]

↳ Finalmente tutti come Stud-lav (NON VA BENE)

[In pratica quando definiamo una GERARCHIA DI TIPI, stiamo definendo dei VINCOLI SULLE ISTANZE. ⊛ → Pag. Succ.

- Per ciò che riguarda la disomogeneità VERTICALE: (TIPI DIVERSI CON VALORE DELLO STESSO ATTRIBUTO)

2) IDENTIFICAZIONE DELLE ISTANZE (di RECORD):

Il modo con cui identifichiamo il RECORD nel modello relazionale è la CHIAVE PRIMARIA.

PROBLEMA RELAZIONALE → Pk → UNICA / "PERSISTENZA"

1. NON UNIFORMITÀ → le chiavi primarie sono diverse da relazione a relazione (chiavi primarie di tipi diversi) della chiave primaria che dipende dalla specifica entità. Non sono in grado di rappresentare tipi diversi nello stesso campo come Pk.
2. NON PERSISTENZA. (Se il mio CF. cambia tutte le relazioni che puntavano al mio CF non puntano più a me).

↳ SOLUZIONE: Utilizzare un identificatore univoco che non dipende dalle caratteristiche dell'istanza dell'entità

[Cioè si usa un SURROGATO: Identificatore OPACO definito dal sistema che identifica un'unica istanza all'interno del sistema. È PERSISTENTE ovvero INVARIABILE per tutta la vita dell'istanza. Tipicamente è un 64 bit (LONG LONG) sufficiente per un indirizzamento automatico del sistema

Da questo punto di vista quello che prima era una chiave primaria ~~ad~~ (CF) adesso diventa una chiave candidata (RIMANE SOLO UN VINCOLO DI UNICITÀ)

4b | Concetti visti: GERARCHIA DI TIPI e IDENTIFICAZIONE TRAMITE SURROGATI

Si trovano sia nell' OBJECT ORIENTED che nell' OBJECT RELATIONAL. Né l' OBJECT ORIENTED né l' OBJECT RELATIONAL risolvono completamente le critiche di Kent.

→ Il problema è che il RECORD viene utilizzato come strumento di modellizzazione

RECORD → AGGREGAZIONE FISICA → (1) Una caratteristica di tipo FISICO viene usata a livello logico. Impatto negativo nelle prestazioni del sistema. Una relazione con 100 attributi in cui accedo sempre a un # limitato di essi, una transazione, dal punto di vista fisico, accede sempre all'intero RECORD (quindi a tutti gli attributi).

Problemi: "SEMANTICI" PART-OF

(2) Altro problema: Un cambiamento nella descrizione dei dati comporta una ristrutturazione nella relazione. (Ristrutturare le TABELLE)

Il concetto di record come aggregazione FISICA non serve a livello logico e può essere dannoso a livello FISICO.

[L'AGGREGAZIONE (Raggruppamento fisico di attr.) è una considerazione FISICA → Non ci sono motivi concettuali per cui 2 attr di una RELAZIONE debbano stare fisicamente in maniera contigua.]

→ L'aggregazione modella una relazione che è fatta di parti. Ma gli attributi spesso non sono una "parte" dell'entità. L'età di una persona è ASSOCIATA ad una persona; non è una PARTE di essa. È un'ASSOCIAZIONE e non un'AGGREGAZIONE (È più un problema filosofico).

[SOL. (SACCO) → Invece di lavorare con le AGGREGAZIONI, lavoriamo con le ASSOCIAZIONI.
[IMP. ID]
[IMP. COGN]
A livello LOGICO vediamo solo ASSOCIAZIONI IRRIDUCIBILI (Relazioni che contengono il minimo # di dati). Con una rappresentazione di questo tipo, a liv. FISICO possiamo mappare RELAZIONI EFFETTIVE come vogliamo
SACCO. The fact model - - INFORMATION SYSTEM 13.1, 1988]

(*) ← Pag. Prec.

Un MODELLO in grado di rispondere alle critiche di KENT deve supportare dei RETICOLI SEMANTICI. Quello che si vuole supportare è il concetto di EREDITARIETÀ MULTIPLA che serve a modellare dei casi che esistono in realtà in maniera semplice. Quindi, invece di avere una GERARCHIA ben definita, possiamo ereditare in maniera multipla da 2 oggetti della gerarchia e definire un nuovo oggetto (es. WORK-STUDENT ottenuto ereditando gli attributi di STUDENT e WORKER). Oltre all'EREDITARIETÀ DEGLI ATTRIBUTI una struttura di questo tipo impone dei vincoli (SUBSUMPTION). È legato a tali vincoli il concetto di ESTENSIONE SUPERFICIALE ed ESTENSIONE PROFONDA (Modo in cui possiamo rappresentare la gerarchia).

Supponiamo di rappresentare la gerarchia come un insieme di:

PERSONE [ ]
STUDENTI [ ]
LAVORATORI [ ]

Consideriamo le entità separate (le PERSONE sono oggetti che sono solo PERSONE: né STUD., né LAVORATORI). Per mantenere il VINCOLO di inclusione (SUBSUMES) uso la diff. tra:

ESTENSIONE SUPERFICIALE: Lo STUDENTE dal punto di vista (intero come soggetto) superficiale sta nell'entità STUDENTE.

Invece, con l'EST. SUP. di PERSONA indichiamo solo le istanze classificate come PERSONA (questo non rappr. la gerarchia) => Il set di istanze definito da PERSONA deve essere definito come: EST. SUP di PERSONA unione l'EST. SUP. di ogni discendente di PERSONA (quindi EST SUP di studenti e lavoratori) Questa è l'ESTENSIONE PROFONDA

Sa

MODELLI AVANZATI: 9/10/2012

MOTIVAZIONI DI ING. DEL SOFTWARE (Nuovi sviluppi dovuti all'ing. del SOFTWARE)

Intorno al '60/'67 nasce il concetto di TDA con SIMULA

ABSTRACT DATA TYPE → IDEA DI INCAPSULAMENTO → "Scatola nera di cui conosciamo INPUT e OUTPUT ma non la rappr. interna" → DEBUG più semplice

- Variabili interne
- Metodi/Funzioni

① Possiamo cambiare la rappr. interna senza cambiare la struttura del progr. in quanto le var. interne sono accessibili solo attraverso i metodi. ② Prototipazione più semplice.

Scelgo delle implementazioni banali per far vedere che il sistema funziona e successivamente raffino tali implementazioni

Intorno al '70 SMALTALK [XEROX PARC] introduce l'idea di GERARCHIA DEI TIPI e quindi l'EREDITARIETÀ di DATI e OPERAZIONI. (Prima gli ADT erano "piatti")

La GERARCHIA DEI TIPI semplifica sia il RIUSO che il DISEGNO INCREMENTALE in quanto permettono il disegno per raffinamenti successivi (quindi ho GERARCHIE di ASTRAZIONE in cui si va specializzando il TIPO facendogli fare un "delta" in più rispetto al PADRE).

Insieme all'EREDITARIETÀ si ha il POLIMORFISMO DEGLI OPERATORI o OVERLOADING che permette di definire la stessa operazione con lo stesso nome su oggetti diversi (l'operazione fa la stessa cosa utilizzando metodi diversi a seconda dell'oggetto) (CIÒ PROVOCA ANCORA UNA SEMPLIFICAZIONE DEL DISEGNO)

Le due cose messe assieme provocano una disciplina di disegno particolare (OO) che ha degli influssi notevoli sulla INTERAZIONE UOMO/MACCHINA

MOTIVAZIONI DI ING. DEL SOFTWARE (L'OBJECT ORIENTED USA QUESTO APPROCCIO)

- IMPEDENCE MISMATCH: la separazione tradizionale tra linguaggi di programmazione e DB dovrebbe essere evitata. Quindi si dovrebbe usare un unico ambiente con dati PERSISTENTI, ossia eliminiamo la parte embedded sul database e consideriamo il DB come avente dei dati (come un ling. di progr.) con la differenza che i dati sono PERSISTENTI.

- APPLICAZIONI CON OGGETTI COMPLESSI (CAD, CAM, GIS, TELECOM, SISTEMI MULTIMEDIALI, MOTORI DI RICERCA, ...): Il problema delle app. di ogg. NON TRADIZ. (gestioni di testo, video, suoni) è che non sono gestiti direttamente dal SIST DI DB. Vogliamo aumentare la capacità del sistema di gestire oggetti che non sono ADT e di gestire oggetti effettivamente COMPLESSI (che richiedono ereditarietà, ecc...)

- PERFORMANCE: I sistemi di DB relazionali sono lenti ma il "PROGRAMMATORE ESPERTO", se usa un ambiente unico con dati persistenti, può programmare l'accesso ai dati in maniera efficiente (USATA COME STRUMENTO DI MARKETING)

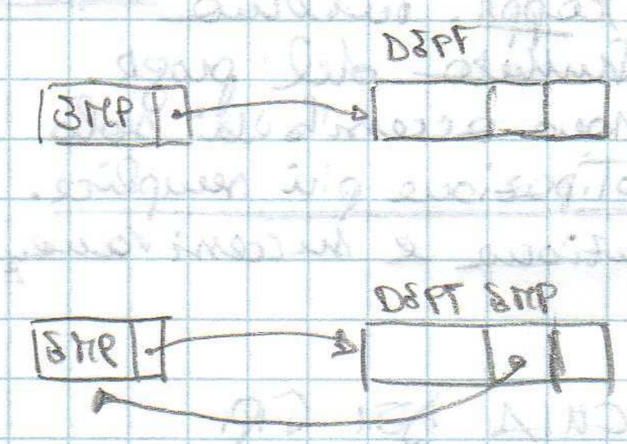
→ Conclusione ovvia dell'usare un unico ambiente con dati persistenti

Alcuni problemi di questo approccio uniforme (OBJECT ORIENTED) sono:

- L'INCAPSULAMENTO: (x l'ING. DEL SOFTWARE è un vantaggio). Su SIST. DI DB il fatto che l'unico modo per accedere ai dati sia attraverso le funzioni ci impedisce di fare tutta una serie di cose. ESEMPI ???

  → Non viene usato nell'ambiente DB

- ASSOCIAZIONI: L'approccio OO però lavora su un <u>modello funzionale</u> in cui le associazioni vengono rappresentate come attributi di oggetti. (Analogamente alla rappresentazione che usiamo nei DB relazionali). Ad es. all'interno dell'opg. EMP ho un riferimento (PUNTATORE) che punta all'opg. DEPT.



  Purtroppo la funzione è da una parte sola e per trovare un EMP a partire dal DEPT dobbiamo definire la funzione inversa in quanto i sistemi OO permettono di avere collezioni di oggetti (si torna ad avere troppi puntatori come nel GERARCHICO e RETICOLARE)

Nel <u>modello TABELLARE</u> (RELAZIONALE) faccio ciò in maniera molto più agevole in quanto con le tabelle definisco in maniera molto più agevole sia la funzione principale che il suo inverso.

Quindi il passaggio al modello funzionale (OO) complica ulteriormente la vita sia perché per ogni funzione bisogna definire l'inversa (con funzioni che hanno molti argomenti diventa piuttosto complicato) sia perché COMINCIO A SPARPAGLIARE I DATI mantenendo sempre un RECORD ma <u>perdendo</u> l'INSIEME DI RECORD (ovvero la TABELLA).
Quindi le informazioni dell'ASSOCIAZIONE diventano PARTE dell'OGGETTO STESSO

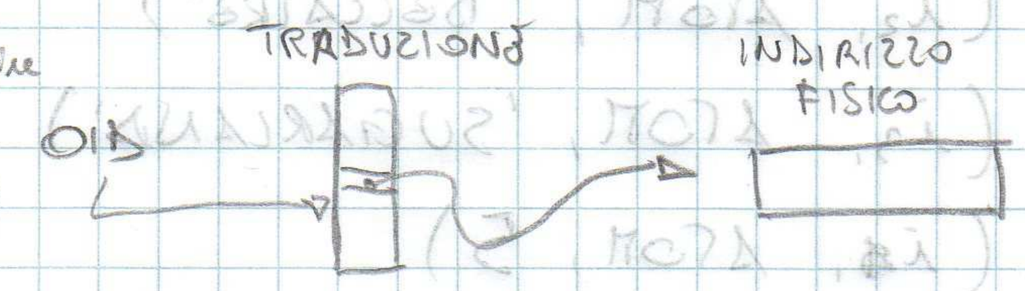
MODELLI AVANZATI: 10/10/2012

CONCETTI FONDAMENTALI DI OODBMS: Standard ODMG 2.0 per OODBMS (CAP.5)

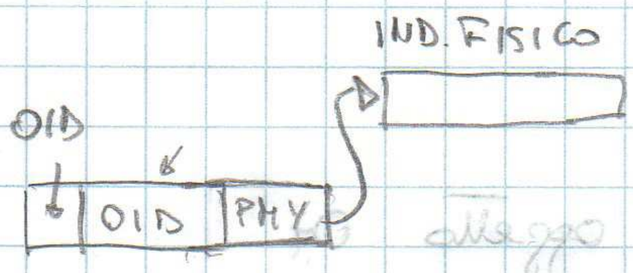
CONCETTO DI IDENTITÀ DELL'OGGETTO:
- Identificativo unico e immutabile;
- indipendente dalle altre caratteristiche dell'oggetto; (NO CF come identificativo)
- Non è solitamente un indirizzo fisico

⇓
"PUNTATORE" (non fisicamente)
VALUTAZIONE MODELLISTICA

L'ID dell'oggetto (OID) viene mappato in un indirizzo fisico tramite una tavola di traduzione (SOLITAMENTE TAVOLA HASH)



Esiste una tecnica, POINTER SWIZZLING (Scambio di puntatori) che, una volta che ha l'indirizzo fisico dell'oggetto, sostituisce o copia l'indirizzo fisico in quello logico (OID diventa l'ind. fisico stesso dell'oggetto). Da quel momento in poi non abbiamo più bisogno della tavola di traduzione ma accediamo direttamente all'indirizzo fisico (in mem. primaria)



C'è una suddivisione tra OGGETTI e VALORI LETTERALI:

OGGETTI: Sono tipicamente complessi
↳ Hanno identificazione tramite OID
- STRUTTURATI: Hanno variabili al loro interno, altre strutture, ecc.
- NON STRUTT.: Sono spazi binari in cui leggiamo delle cose (BLOB - Binary Large Object)

Permettono di gestire in maniera non interpretata (IN BINARIO) degli oggetti COMPLESSI che verranno poi gestiti con metodi specifici come immagini, video, testi, ecc

VALORI LETTERALI: Non hanno OID e hanno solo il VALORE

Gli OGGETTI sono costruiti da altri OGGETTI o VALORI mediante l'applicazione di COSTRUTTORI DI TIPI

I TIPI che utilizziamo per COSTRUIRE gli oggetti sono PREDEFINITI:

Oggetto Atomico ATOM VALORE ATOMICO

COLLEZIONI (Raggruppamenti di oggetti)
- SET → INSIEME
- LIST → INSIEME ORDINATO
- BAG → MULTIINSIEME
- ARRAY → VETTORE

OID

Sono collezioni di OID = Permettono di mettere insieme gli oggetti in un SET, in una LIST, ecc

Un altro TIPO è TUPLE che è un AGGREGATO DI OID → Come una Tupla nel SISTEMA RELAZIONALE
(TAPOL in inf)

TIPO STRUTTURATO TUPLE O_1, OID, OID, OID

Es.: Identifico un oggetto con una terna
(OID, COSTRUTTORE, VALORE)

Ad es. O_1 (i_1, ATOM, 'HUSTON')
O_2 (i_2, ATOM, 'BELCAIRE')
O_3 (i_3, ATOM, 'SUGARLAND')
O_4 (i_4, ATOM, 5)
O_5 (i_5, ATOM, 'RICERCA')
O_6 (i_6, ATOM, '1988-05-22')

Costruzione di oggetti

Abbiamo definito tutta una serie di valori atomici

Comincio ad aggregare questi valori e definisco un oggetto O_7

O_7 (i_7, SET, {i_1, i_2, i_3})] Ho fatto una COLLEZIONE di 3 ogg. definiti prima
↳ COSTRUTTORE — O_7 è un oggetto che fa rif. ad altri oggetti

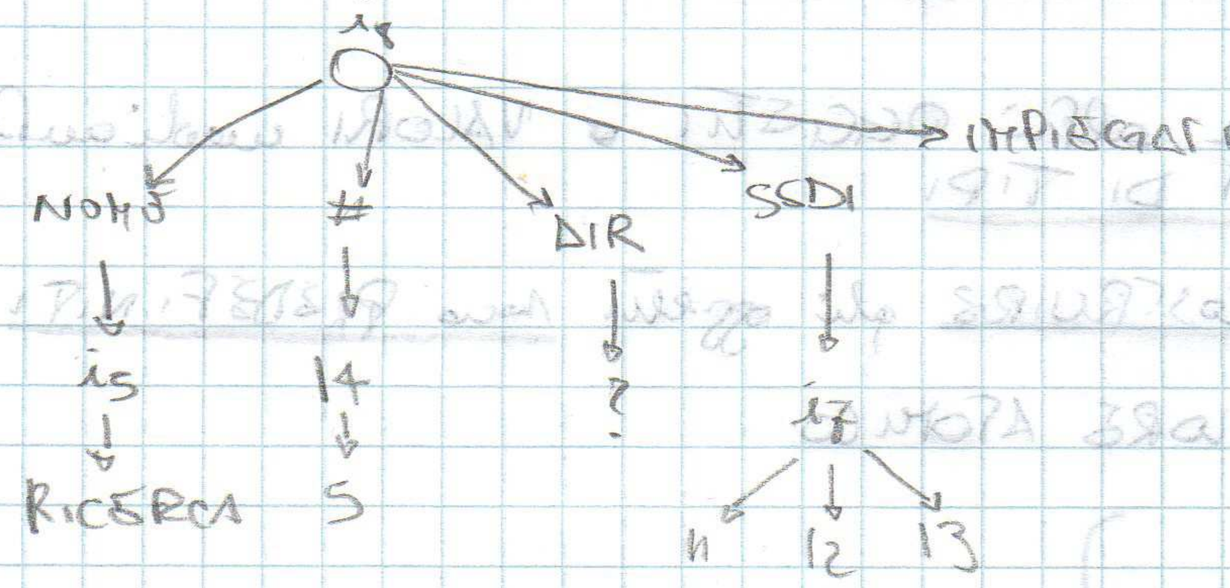
Creo un nuovo oggetto O_8

Devono ancora essere definiti

O_8 (i_8, TUPLE, <NOME_D: i_5, N-DIP: i_4, DIR: i_9, SEDI: i_7, IMPIEGATI: i_{10}>)
↳ Questo oggetto è di TIPO TUPLE

Abbiamo definito un oggetto con una str. complessa partendo da COSTRUTTORI DI OGGETTI. Tutti questi oggetti sono identificati da un OID e sono indirizzabili da altri oggetti. Con i VALORI ATOMICI e le COLLEZIONI dal punto di vista semantico non definito nulla. Con l'utilizzo delle TUPLE definisco la SEMANTICA.

Ho così definito una struttura a grafo che parte da O_8



Con l'applicazione dei vari TIPI di COSTRUTTORI abbiamo definito una struttura a GRAFO in cui possiamo trovare i valori navigando il grafo.

I COSTRUTTORI PREDEFINITI, come detto, costruiscono un oggetto mettendo assieme VALORI ATOMICI o COLLEZIONI o VALORI STRUTTURATI

7a

In questo contesto diventa complesso il concetto di:

UGUAGLIANZA TRA 2 OGGETTI:

Dati 2 oggetti i_k e i_j posso avere 2 tipi di uguaglianza:

UGUAGL. PROFONDA: in cui si richiede la stessa struttura, gli stessi OID dei grafi allo stesso punto della struttura e quindi gli stessi valori (GRAFI ESATTAMENTE UGUALI)
- STESSO STATO
- STESSA STRUTTURA
- STESSI OID

UGUAGL. SUPERFICIALE: Rilassa il tipo di ug. precedente concentrandosi
(MENO RESTRITTIVA) sullo stato:
- STESSA STRUTTURA
- STESSI VALORI
- OID DIVERSI

Es. Suppongo di avere 3 oggetti TUPLE e 3 oggetti ATOMI:

O_1 (i_1, TUPLA, < a_1 : i_4 , a_2 : i_6 >) → questa è un'associazione UNARIA tra oggetti

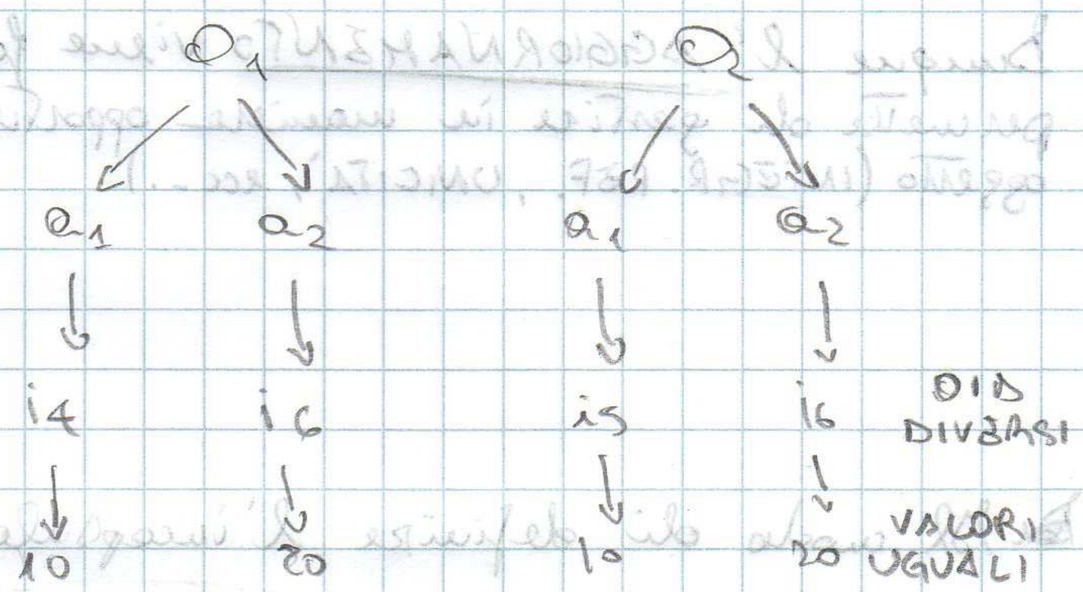
O_2 (i_2, TUPLA, < a_1 : i_5 , a_2 : i_6 >)

O_3 (i_3, TUPLA, < a_1 : i_4 , a_2 : i_6 >)

O_4 (i_4, ATOMO, 10)

O_5 (i_5, ATOMO, 10)

O_6 (i_6, ATOMO, 20)



→ Ci sono OID diversi

O_1 e O_2 non hanno ug. profonda. Se i_4 e i_5 hanno gli stessi valori (come avviene in questo esempio) ⇒ O_1 e O_2 hanno ug. superficiale

[All'interno di questa struttura gli OID usati da altri OGGETTI rappresentano delle ASSOCIAZIONI tra oggetti diversi

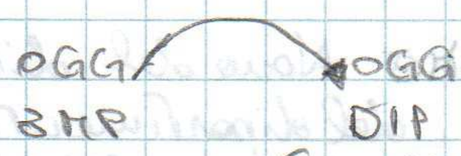
L'ASSOCIAZIONE BINARIA (tra 2 oggetti) può possedere un riferimento INVERSO che viene definito esplicitamente.

Es. Ho una tupla relativa agli impiegati:

O_1 (i_1, TUPLA, < EMP_NAME : 12 , EMP_DIP : i_3, ___ >)

O_2 (i_2, ATOM, "ROSSI")

O_3 (i_3, ___)



Associazione tra l'oggetto relativo all'impiegato e l'oggetto relativo al dipartimento

↳ In questo caso definisco un riferimento ad un oggetto che rappresenta un'altra ENTITÀ. Stiamo definendo un'ASSOCIAZIONE BINARIA per cui un impiegato LAVORA in un DIPARTIMENTO. Ma in realtà questa ASSOCIAZIONE ha un'unica direzione. Per avere quindi EFFETTIVAMENTE UN'ASSOCIAZIONE BINARIA siamo costretti a definire esplicitamente delle funzioni inverse (COME NEL MODELLO RETICOLARE con la differenza che il LIVELLO FISICO rimane del tutto trasparente)

(INFORMATION HIDING)
Fb **INCAPSULAMENTO**. Quello che implica la definizione di ADT è un INCAPS. TOTALE in cui tutte le propr. sono invisibili direttamente ed accessibili attraverso operazioni definite all'interno dell'ADT stesso.

Dal punto di vista dei sistemi di DB è altamente inefficiente perché abbiamo bisogno di un accesso diretto agli attributi definiti all'interno delle strutture per permettere di avere un accesso ai dati più VELOCE

Questo approccio viene quindi rilanciato facendo una separazione tra gli attributi che possono essere in parte visibili e in parte nascosti.

STATO (Gli attr. possono essere) [PROPRIETÀ/ATTRIBUTI
- VISIBLI
- NASCOSTI

Tipicamente sono visibili in lettura e gestite in scrittura dai METODI

OPERAZIONI [INTERFACCIA/SIGNATURE: Prototipo della funzione
METODO/IMPLEMENTAZIONE: Implementazione della dichiarazione della funzione

Se abbiamo solo la dichiarazione abbiamo una CLASSE ASTRATTA che non è ISTANZIABILE

Dunque l'AGGIORNAMENTO viene fatto in maniera incapsulata perché mi permette di gestire in maniera opportuna gli eventuali vincoli che ho lo specifico oggetto (INTEGR. REF., UNICITÀ, ecc..)

Il modo di definire l'incapsulamento è la definizione di CLASSE:

CLASSE: DEFINIZIONE DI TIPO / OPERAZIONI

Si utilizza la DOT NOTATION per indirizzare variabili o operazioni di uno specifico oggetto
Per es. se ho un oggetto D posso usare -d.età

-d.aumentaStipendio(10000)

Questa notazione può essere estesa per comporre tra di loro dati e operazioni ottenendo la PATH EXPRESSION (espressione di cammino)

Es. EMP < DIP, NOME >
→ DIP < NOME, DIR > → EMP

Con la DOT NOTATION e.NOME restituisce il nome dell'impiegato

Posso comporre le cose scrivendo ad esempio e.DIP.DIR.NOME Nome del direttore del dipartimento in cui lavora l'impiegato specifico 'e'

Se termini relazionali stiamo definendo dei JOIN in maniera molto più naturale rispetto a quanto lo facciamo in termini relazionali

CONCETTO DI PERSISTENZA:

Non tutti gli oggetti sono persistenti, che servono per la logica del programma
↳ OGGETTI TRANSIENTI.

OGGETTI PERSISTENTI: Rendiamo persistenti gli oggetti attraverso 2 meccanismi:

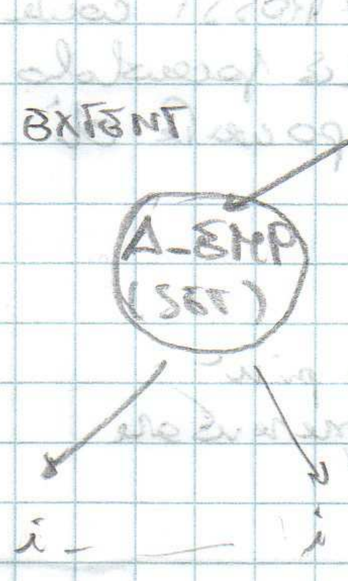
① Per NOME (NAMING): Diamo un nome agli oggetti; il DB "ricorda" questo nome e attraverso di esso mi permetterà di accedere agli oggetti. Non è un metodo "semplice" in quanto possiamo dare un nome ad un # limitato di oggetti ma non possiamo dare un nome a tutti gli ogg. che voglio rendere persistenti.

② Per RAGGIUNGIBILITÀ (REACHABILITY): Definita nel seguente modo:

B è RAGGIUNGIBILE da A se ∃ una seq. di riferimenti che da A porta a B. (Nell'es. precedente il DIPARTIMENTO è ragg. da IMPIEGATO)
Tale concetto è usato nel seg. modo:

SE A È PERSIST ⇒ Ogni B RAGGIUNGIBILE da A è PERSISTENTE

① e ② Ulteriori assieme ~~ai~~ permettono di avere una certa flessibilità:



Posso definire con un NOME un oggetto persistente (ad es. il NOME che do al DB). Tutto ciò che è raggiungibile dall'ogg. a cui ho dato un NOME diventa persistente.

Il DB è una collezione di altre collezioni, ognuna delle quali mantiene oggetti di una specifica classe. In questo caso diamo i nomi a DB, IMPIEGATI, DIPARTIMENTI ecc. rendendoli PERSISTENTI. Tutto ciò che è raggiung. da loro è PERSISTENTE. In pratica abbiamo usato lo stesso # di NOMI che avremmo utilizzato con un DB relazionale.

I NOMI che usiamo in questo modo sono in numero limitato e sono di uguale # rispetto a un DB relazionale. Stiamo, in pratica, SIMULANDO DELLE RELAZIONI. A-EMP corrisponde (nel mod. rel.) alla rel. che contiene tutti gli impiegati.

Questo meccanismo che permette di raggruppare logicamente gli oggetti è viene chiamato EXTENT e fa parte dello STANDARD ODMG.

8b CONCETTO DI GERARCHIA DI TIPI con EREDITARIETÀ già visti precedentemente.

(torno indietro)

OGGETTI COMPLESSI:

Differenza tra OGGETTI / VALORI LETTERALI (Nomi, stipendi, date di nascita, ...) (Elimino gli OID dagli oggetti che non ne hanno bisogno)

OGGETTI: Ha una vita propria

VALORI LETTERALI: È associato ad altri oggetti

OWNERSHIP SEMANTICS: vuol dire che i valori letterali sono incapsulati all'interno dell'oggetto padre e dipendono da lui

↳ Se distruggo l'oggetto padre distruggo anche tutti i valori letterali legati a lui

Gli VAL LETTERALI HANNO

Gli OID hanno invece una:

REFERENCE SEMANTICS: l'componente oggetto INDIPENDENTE riferito tramite il suo ID

Es. EMP

(i1, TUPLA, <NOME: i2, DIP: i3>)

(i2, STRING, "ROSSI") → È inefficiente. Posso rappresentare "ROSSI" come un VALORE LETTERALE facendolo diventare una componente di un altro oggetto.

(i3, TUPLA, <NOME_D: i4, ...>)

Al sistema diventa

(i1, TUPLA, <NOME: "Rossi", DIP: i3>) questo è un modo più EFFICIENTE di rappresentare le cose.

(i3, TUPLA, ~~<NOME~~ ...)

(OBJECT DATA MANAGER GROUP)

STANDARD ODMG 2.0 è composto da 4 parti:

- MODELLO AD OGGETTI → Concetti che usiamo per la modellizzazione
- LINGUAGGIO DI DEFINIZIONE (ODL) OBJECT DEFINITION LANGUAGE
  ↳ Sintassi con cui definiamo i vari costrutti
- LINGUAGGIO DI INTERROGAZIONE (OQL) OBJECT QUERY LANGUAGE
  ↳ Specifica come vengono implementate le interrogazioni non procedurali
- BINDING A LINGUAGGI DI PROGRAMMAZIONE (OO) OBJECT ORIENTED
  ↳ Specifica come i costrutti ODL e OQL vengono mappati nei ling. di prog (C++, JAVA, ...)

→ Definizione dello schema indipendente dal linguaggio di programmazione →

MODELLO A OGGETTI: (STANDARD ODMG) → Abbiamo una distinzione tra:
2.0 → Standard ODBMS

OGGETTI
Descritti da:
OID - unico nel sistema
NOME - [OPZIONALE]
TEMPO DI VITA → Persistente / Transitorio
STRUTTURA → Collezione / Atomico
→ Permette di accedere ad uno specifico oggetto
→ Tutto ciò che non è considerato collezione è ATOMICO (anche una STRUCT in questo caso è un valore ATOMICO)

VALORI LETTERALI (Sono parte dell'ogg.)
VALORE (Conservano solo il valore)
NON HANNO ID
→ Possono essere:
VALORI ATOMICI (long, short, ...)
STRUTTURATI STRUCT
COLLEZIONI → OGGETTI / → VALORI

16/10/2012

Oltre a questi concetti di base abbiamo dei concetti di modellizzazione:

INTERFACE → DEFINIZIONE DI TIPO ASTRATTA
↳ Definiamo le operazioni che il tipo esporta ma, l'INTERFACE NON È DIRETTAMENTE ISTANZIABILE

Interface Object → Interfaccia primitiva: Definisce i metodi, visibili dall'esterno, che permettono di lavorare sull'oggetto
{ boolean same_as (in Object o); IDENTITÀ - Funzione di comparazione con UGUAGLIANZA PROFONDA
Object copy (); Copia dell'oggetto
void delete (); } Distruzione dell'oggetto
→ Tutti gli oggetti derivano da questa definizione

La Object Factory permette di localizzare la costruzione degli oggetti invece di distribuirla nell'intero sistema
→ Gli oggetti vengono costruiti tramite le OBJECT FACTORY

Le INTERFACE NON SONO ISTANZIABILI MA SONO EREDITABILI
È POSSIBILE L'EREDITARIETÀ MULTIPLA

È possibile creare un'altra interfaccia ereditando da più interfacce i metodi disponibili (EREDIT. MULTIPLA).
L'interface serve per definire l'infrastruttura del sistema e quindi derivando tutto dall'interface object abbiamo dei metodi base che sono applicabili a qualunque oggetto. L'infrastruttura rappresenta un livello di disegno che non contiene ancora nulla. Sarà poi di disegno l'appl. vera e propria (il DB ADMINISTRATOR) a definire delle classi

MODELLO A OGGETTI: (STANDARD ODMG) → Abbiamo una distinzione tra:
2.0 → Standard ODBMS

OGGETTI
Descritti da:
OID - unico nel sistema
NOME - [OPZIONALE]
TEMPO DI VITA → Persistente / Transitorio
STRUTTURA → Collezione / Atomico
→ Permette di accedere ad uno specifico oggetto
→ Tutto ciò che non è considerato collezione è ATOMICO (anche una STRUCT in questo caso è un valore ATOMICO)

VALORI LETTERALI (Sono parte dell'ogg.)
VALORE (Conservano solo il valore)
NON HANNO ID
→ Possono essere:
VALORI ATOMICI (long, short, ...)
STRUTTURATI STRUCT
COLLEZIONI → OGGETTI / → VALORI

16/10/2012

Oltre a questi concetti di base abbiamo dei concetti di modellizzazione:

INTERFACE → DEFINIZIONE DI TIPO ASTRATTA
↳ Definiamo le operazioni che il tipo esporta ma, l'INTERFACE NON È DIRETTAMENTE ISTANZIABILE

Interface Object → Interfaccia primitiva: Definisce i metodi, visibili dall'esterno, che permettono di lavorare sull'oggetto
{ boolean same_as (in Object o); IDENTITÀ - Funzione di comparazione con UGUAGLIANZA PROFONDA
Object copy (); Copia dell'oggetto
void delete (); } Distruzione dell'oggetto
→ Tutti gli oggetti derivano da questa definizione

La Object Factory permette di localizzare la costruzione degli oggetti invece di distribuirla nell'intero sistema
→ Gli oggetti vengono costruiti tramite le OBJECT FACTORY

Le INTERFACE NON SONO ISTANZIABILI MA SONO EREDITABILI
È POSSIBILE L'EREDITARIETÀ MULTIPLA

È possibile creare un'altra interfaccia ereditando da più interfacce i metodi disponibili (EREDIT. MULTIPLA).
L'interface serve per definire l'infrastruttura del sistema e quindi derivando tutto dall'interface object abbiamo dei metodi base che sono applicabili a qualunque oggetto. L'infrastruttura rappresenta un livello di disegno che non contiene ancora nulla. Sarà poi di disegno l'appl. vera e propria (il DB ADMINISTRATOR) a definire delle classi

CLASS → USATE PER LE DICHIARAZIONI DELLE CLASSI/TIPI DEFINITI DALL'"UTENTE" (l'utente in questo caso è) ← In questo caso chi disegna il DB è chi disegna il DB (DB ADMINISTRATOR)

Rappresenta la definizione di una tupla nel SIST RELAZIONALE

Se ad es. vogliamo definire un TIPO PERSONA useremo un meccanismo di CLASS che in generale EREDITA le interfacce da altre definizioni e aggiunge dei DATI all'interno (I DATI possono essere collezioni, strutture, valori letterali).

L'indirizzamento dello STATO dell'oggetto viene implementato con:

DOT NOTATION → O1 = O.copy();

ARROW NOTATION → O1 = O→copy();

Un'altra componente importante del MODELLO sono le:
COLLEZIONI → DERIVATE TUTTE DALLA INTERFACE COLLECTION

```
interface Collection : Object
{ exception ElementNotFound (any element);   Meccanismo di gestione degli errori
    unsigned long cardinality();   → Cardinalità
    boolean is_empty(); }
```

Le collezioni che abbiamo a disposizione sono quelle viste in precedenza, ovvero:

SET → insieme senza duplicati

LIST → insieme ordinato con duplicati

BAG → Multiinsieme: possibilità di valori duplicati

ARRAY → Vettore

DICTIONARY → < K, V > Memoria associativa che permette di trovare un valore data la chiave di ricerca (Tavola Hash

Queste interfacce hanno i metodi di "interface collection" e una serie di metodi specializzati che dipendono da ciò che l'interfaccia fa.

Il modo con cui si istanziano queste collezioni è il seguente:

SET < STUDENT > Una collezione definita in questo modo potrà contenere sotto forma di insieme senza duplicati solo degli oggetti di tipo studente

→ Una definizione di questo tipo (FORTE) impedisce la costruzione di CATEGORIE.

10a

MODELLI AVANZATI

16/10/2012

→ viene definita dall'"utente" (DB ADMINISTRATOR)

OGGETTO ATOMICO → Non Collezione

Class <NOME> [EXTENDS <TIPO>] → Opzionalmente posso ereditare i metodi di un'altra classe. Questa ereditarietà è singola → → Possiamo ereditare da 1 solo tipo → Ereditarietà SINGOLA

Vengono resi PERSISTENTI

EXTENT → Collezione della classe che mantiene tutti gli oggetti della classe: è equiv. alla relazione nel mod. rel

KEY → Chiavi candidate → Serve per definire i VINCOLI DI UNICITÀ

{ ATTRIBUTE OPPURE RELATIONSHIP → Att. normale che può essere struttura, valore atomico, etc etc...

OPERAZIONI { <metodi> + <exception

→ Rappresenta uno dei punti di ingresso nel DB

Es. CLASS PERSON
{EXTENT Persons SET<PERSON>
↳ Viene automaticamente popolato nel momento in cui creo oggetti di tipo persona

Quindi permette di definire un iteratore sull'estensione della classe e di enumerare tutti gli elementi della classe (come quando nel mod. rel. accediamo ad una relazione e ne leggiamo gli elem)

Mette in relazione gli oggetti appartenenti a 2 classi. In pratica definiamo un associaz. binaria

Es. di RELATIONSHIP

Class Dipartimento
{ ; Serie di i attributi

relationship SET< Corpo_docente > ha_corpo_docente 38.00

inverse → Permette di definire la rel. inversa

Corpo_docente :: lavora_in

↳ Approccio FUNZIONALE

→ Definisco una relazione 1:N da cui restituisce un insieme di oggetti "professori" mediante una definizione di propr. "ha_corpo_docente"

Tramite d.ha_corpo_docente trovo l'insieme di prof. che appartengono al dipartimento d questa rel però è solo in un verso ⇒
⇒ Dal DIP posso risalire ai PROF

Con la keyword INVERSE definisco la funzione inversa ⇒
⇒ Da "corpo_docente" arrivo a DIPART. usando "lavora_in"

Ad es.

(d.ha_corpo_docente)

Restituisce l'insieme (N) dei Prof di un DIP

(d=t.lavora_in)

Restituisce il (1) DIP in cui lavora t

Ho modellato una rel. 1:N. Le rel 1:1 e N:N possono essere modellate allo stesso modo

11a

MODELLI AVANZATI DI DB

06/11/2012

REVIEW

OO / OR

- MODELLISTICO → ER (SQL 99)
  - GERARCHIE (LATTICES) DI TIPI
  - IDENTIFICAZIONE TRAMITE SURROGATI (OID)
  - CATEGORIE

  (SQL 99)

- SO → OO (ODMG)
  - GERARCHIE DI TIPI
  - ADT

  } OQL — NESSUNA COMPATIBILITÀ COL PASSATO

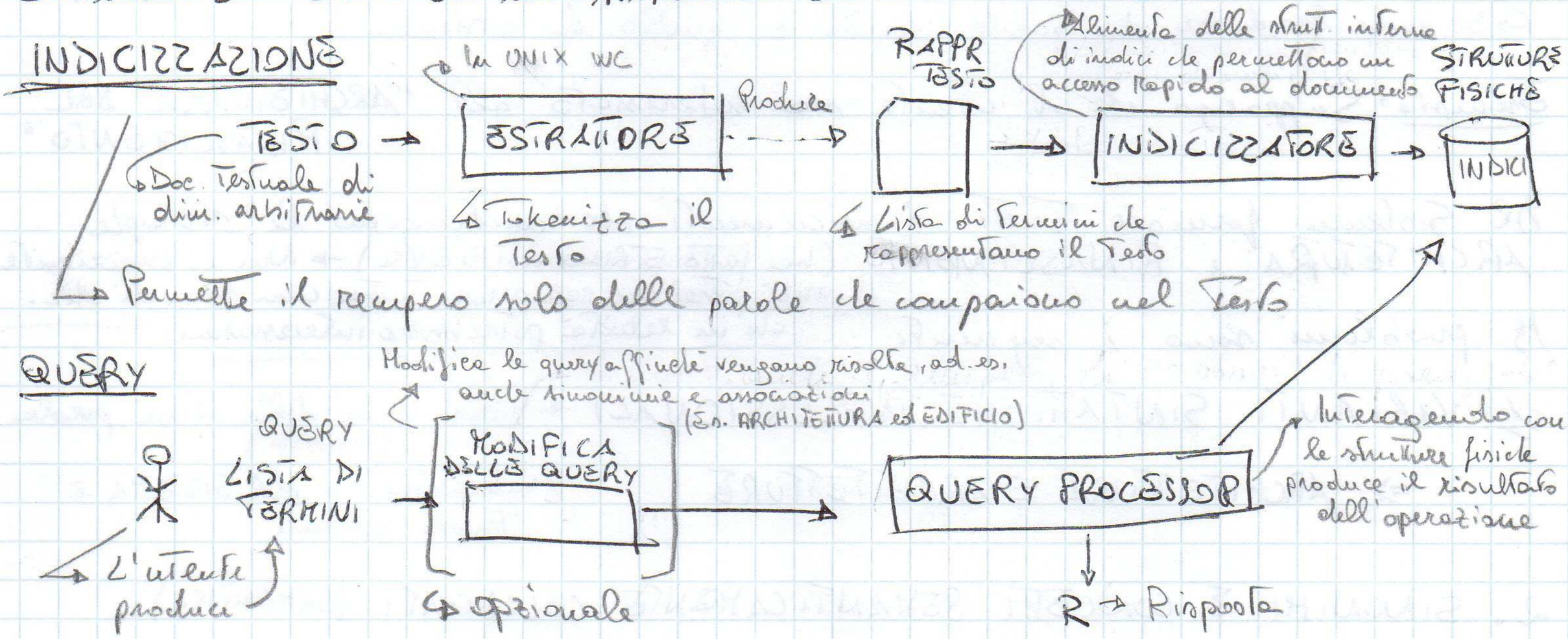
Lettura: THE FACT MODEL: Information System 1988

IR - TEXT RETRIEVAL - MOTORI DI RICERCA

Primi lavori negli '50
Commercializzati '80

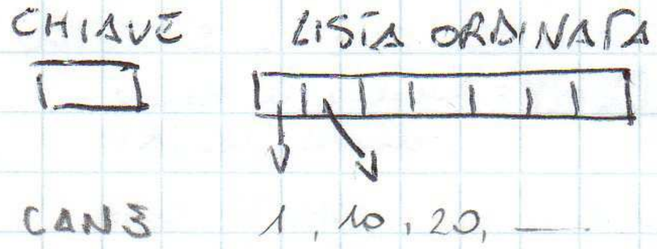
Schema di base di un sistema di TEXT RETRIEVAL

INDICIZZAZIONE



Tale struttura funziona bene se cerco una stringa di caratteri (come un nome proprio) ma funziona male se cerco un concetto. (Architettura Rinascimentale)

Le STR. FISICHE sono generalmente gestite con LISTE INVERTITE (è un meccanismo per la gestione di indici non clusterizzati)



Le LISTE INVERTITE hanno una CHIAVE e una LISTA che contiene i puntatori agli oggetti relativi a quella chiave. Nel nostro caso abbiamo una CHIAVE (Cane) e una LISTA di identificativi di documenti che contengono il termine della CHIAVE

Queste LISTE INVERTITE sono gestite a loro volta da una struttura ad indice ⇒ Se un utente ha immesso il termine "cane" nella query, riceverà in risposta i titoli dei documenti indicizzati dalla lista ordinata (INVERTITA) che ha come chiave il termine CANE.

11b

Dopo che il doc. è stato INDICIZZATO, questo serve poi solo come risultato di una query => Può non esistere più o essere fuori linea.

Le indicazioni sulle strutture interne rimangono ma il doc. può scomparire.

Le strutture a LISTE INVERTITE è particolarmente adatta per risolvere delle query di tipo BOOLEANO => A fronte di una specifica interrogazione booleana, lavorando solo ed esclusivamente sulle strutture di indici (nel nostro caso le LISTE INVERTITE) sono in grado di trovare un risultato efficientemente.

PROBLEMA: Enorme GAP semantico tra il modello del sistema e il modello dell'utente
↳ Si basa sulle stringhe
↳ Si basa sui CONCETTI

GAP SEMANTICO

SISTEMA → String

USER → Concetti

↳ Si vuole tradurre da questo livello a quello di sistema

Esempio: Suppongo che l'utente sia interessato all'"ARCHITETTURA DEL RINASCIMENTO"

Il Sistema fornisce tutti i documenti che contengono le stringhe ARCHITETTURA e RINASCIMENTO (ho fatto STRING RETRIEVAL) → Non è esaustivo perché potrei non recuperare tutta una serie di doc. che in realtà potrebbero interessarmi

I problemi sono i seguenti:

1. VARIANTI SINTATTICHE INFLESSIONALI → Varie forme della stessa parola (SINTASSI)

   => ARCHITETTURA ≡ ARCHITETTURE — L'interpretazione del SISTEMA è troppo letterale

2. SINONIMI E CONCETTI SEMANTICAMENTE COLLEGATI (SEMANTICA)

   ARCHITETTURA → "Edificio Rinascimentale" è collegato ad "Architettura del Rinascimento" così come sono collegate allo stesso concetto pure altre cose come "Chiesa Rinascimentale"

   ↙ EDIFICIO ↘

   CHIESA — FORTE

   Questa parte richiede dei meccanismi complicati per TRADURRE QUELLO CHE L'UTENTE STA CHIEDENDO IN QUALCOSA CHE IL SISTEMA PUÒ CAPIRE ⟶ MODIFICA DELLA QUERY = Amplia il senso della QUERY

3. TEMATICA È IGNORATA NEL DOCUMENTO → Es. I manuali UNIX non citano mai il termine UNIX

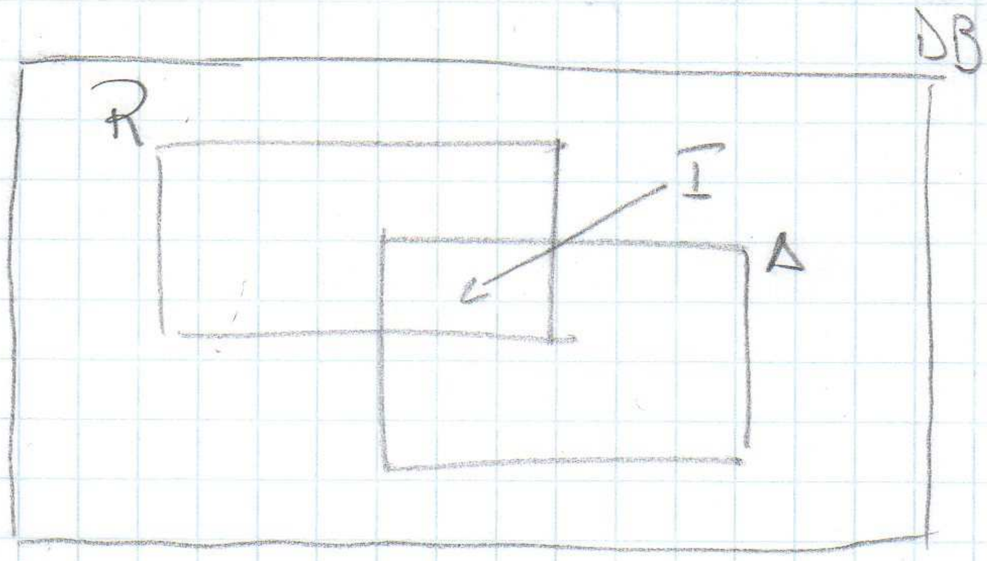
↳ Ad es. dato che il termine "Rinascimento" è stato coniato dopo => tutti i documenti che sono stati scritti in quel periodo non conterranno tale termine.

Le MISURE che vengono usate normalmente in Information Retrieval per la VALUTAZIONE del RISULTATO sono le seguenti:

RECALL (ESAURIENZA): Capacità del sistema di recuperare tutti i doc. rilevanti rispetto all'interrogazione
↳ Quantità di doc. ril. recuperati rispetto alla tot. dei doc. recuperati

PRECISION (ASSENZA DI RUMORE): Misura la presenza di doc. non rilevanti nel risultato (RUMORE).

Rappresentazione grafica:



R = doc. Rilevanti

A = Risultato (ANSWER)

I = R ∩ A

Si può definire in maniera normalizzata la QUALITÀ DELLA RISP. in termini di ESAURIENZA e PRECISIONE:

$$\text{RECALL} = \frac{|I|}{|R|} \leq 1$$

→ Se RECALL = 1 ⇒ Ho recuperato tutti i documenti rilevanti rispetto al mio interesse
↳ Assenza di perdita di doc. rilevanti

$$\text{PRECISION} = \frac{|I|}{|A|} \leq 1$$

→ Se PRECISION = 1 ⇒ Ho totale assenza di Rumore

Noi vorremmo avere RECALL = PRECISION = 1 per recuperare tutti e solo i doc. rilevanti
Questa è una valutazione SOGGETTIVA → Può essere fatta da un esperto che conosce il DB

Dato il GAP SEMANTICO descritto prima si ha che i nostri sistemi tendono ad avere BUONA PRECISION ma PESSIMO RECALL in quanto quest'ultimo dipende dal GAP tra il modello dell'utente e il modello del sistema.

Oltre a perdere in ESAURIENZA per i motivi appena descritti, il linguaggio di interrogazione ci può portare a una perdita di ESAURIENZA possibilmente controbilanciata da un aumento di PRECISIONE →

→ Ad esempio quando facciamo QUERY IN AND per ridurre un risultato troppo ampio (Se ho un ris. di 1 milione di doc ⇒ lo metto in AND con qualcosa e riduco la dim. del ris.)

Questo (METTERE IN AND) può portare alla PERDITA DI DOC. RILEVANTI

Es.: Se l'utente chiede ADAM (Adamo biblico) ottiene in risp. tutti gli ADAM presenti nel DB (quindi non solo i doc. inerenti all'Adamo biblico).
Abbiamo quindi un RUMORE ESTREMAMENTE ELEVATO
Se faccio la QUERY in and con ADAM AND EVA miglioriamo di molto la precisione ma essendo che EVA è nata dopo di ADAMO perdiamo tutti i doc. (rilevanti) in cui vi è solo ADAM ⇒ DIMINUISCE IL RECALL.

TERMINE: Token, stringa che estraiamo dal testo

WHITE LISTS → Vocabolario controllato → Standardizzazione della rappresentazione del documento. Nel vocabolario controllato possiamo mettere dei concetti (classificati da un utente - CLASSIFICAZIONE PER KEYWORDS)
(COSTOSO)

BLACK LISTS → Termini NON INDICIZZATI (STOP WORDS). Sono termini troppo generali per avere un'utilità dal punto di vista del reperimento (come gli articoli).
← Diminuzione di strutture di INDICI

3ª MOD. AV. 07/11/2012

→ è ciò che decidiamo di INDICIZZARE

TERMINE: cosa usiamo come termine?

- In prima approssimazione usiamo le stringhe del documento

NON TUTTI I TERMINI DEVONO ESSERE MEMORIZZATI (articoli, voci verbi servili, etc sono irrilevanti dal punto di vista del recupero dei documenti)

Tali termini (possono essere IGNORATI) vengono detti

STOP WORDS / BLACK LIST → Bastano poche STOP WORDS per avere un notevole risparmio perché la distrib. delle parole (come frequenza) non è uniforme ma segue la LEGGE DI ZIPF LAW

→ Servono per ottenere un risparmio notevole nelle strutture di indice

LEGGE DI ZIPF LAW

$$P_i = \frac{C}{i} \qquad \sum_{i=1}^{N} P_i = 1$$

C → Cost. di normalizzazione

P_i → prob. del TERMINE i-esimo

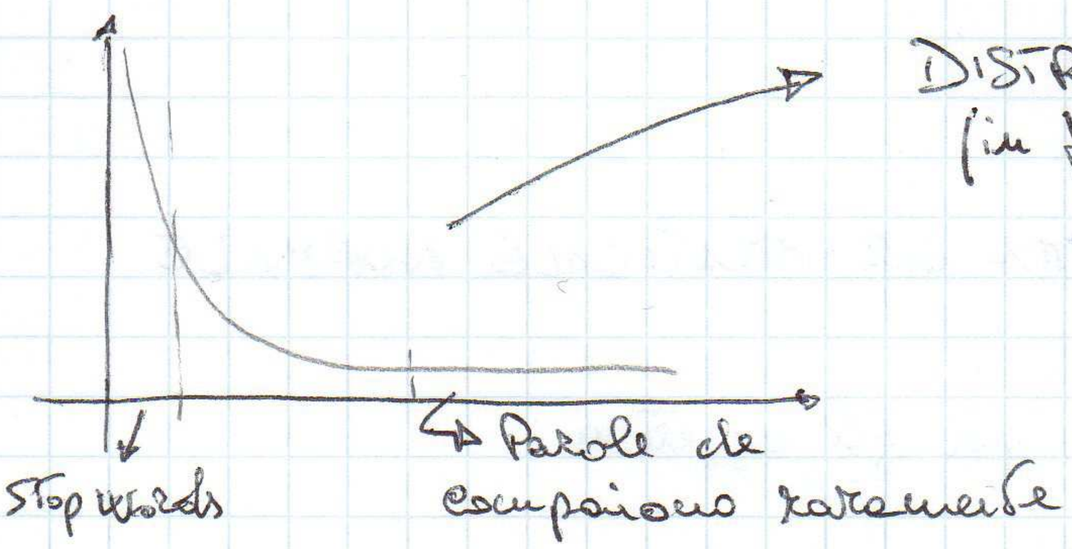
Essendo delle prob. devono essere normalizzate a 1

$$C = 1 + \frac{1}{2} + \dots + \frac{1}{N} = \sum \frac{1}{k} = H_N \simeq \ln(N) + \gamma$$

H_N → Numero Armonico di ordine N

in cui $\gamma = 0{,}577$ → NUMERO DI EULERO

Tale legge è di tipo $\frac{1}{x}$



DISTRIBUZIONE DELLE PAROLE NEI TESTI (in frequenza)

Tale legge si verifica in pratica in molte altre situazioni (ad. es. questa legge regola la distrib. della ricchezza, distr. della popolazione, etc...).

Problemi: e tecniche per la diminuzione del GAP SEMANTICO

1. TOKENIZZAZIONE (cosa usiamo come separatori? In base a ciò che facciamo, certi termini non sono ricercabili)

Tale delicata che può comportare perdita di RECALL

2. STOP WORDS

Anche queste possono comportare perdita di PRECISION

Ad. es. il termine "di". Se lo considero come STOP WORDS non riuscirò a tokenizzare, per es., quelle parole che contengono il termine "di" come il cognome "Di Pietro" → In questo caso avrei come risp. tutti i doc. che contengono la parola "Pietro" → PERDITA DI PRECISION

3. INFLESSIONI

Per aumentare l'ESAURIENZA vogliamo considerare tutte le varianti grammaticali di un termine, come unico termine

Si può fare in 2 modi

- RIDUZIONE A TEMA DELLA PAROLA (STAMMING) → FUNZIONA MALE

↳ Invece di considerare la parola CANE, considero CAN

Problemi:

① Il tema non necessariamente rappresenta univocamente una serie di inflessioni che hanno lo stesso significato.

Ad esempio il tema NEUTR va bene per NEUTRO, NEUTRALE, NEUTRALITÀ, ecc. ma va male per NEUTRONE → Se cerco NEUTRONE ottengo anche i doc contenenti i termini NEUTRO, NEUTRALE, ecc.

[Ciò che si verifica in questo caso (STIAMO CERCANDO DI MIGLIORARE IL RECALL) ⇒ otteniamo perdita di PRECISION (questa non è una regola matematica)

② Verbi IRREGOLARI

Ad es. il verbo ANDARE ha temi diversi - "Vado" e "andrò" non hanno lo stesso tema

- NORMALIZZAZIONE GRAMMATICALE

Vogliamo normalizzare ad una FORMA GRAMMATICALE NORMALE scelta a priori

Ad es. FEMMINILE PLURALE per gli aggettivi

INFINITO per i verbi

VANTAGGI: Aumento dell'ESAURIENZA, Diminuzione delle strutture interne → perché diminuisce il # di termini

SVANTAGGI: In certe situazioni, anche in questo caso, si verifica un AUMENTO DI RUMORE

Ad es.

UOMO

DONNA

4. SINONIMI E TERMINI CORRELATI

La sinonimia e i termini correlati vengono gestiti attraverso:

THESAURUS: Tradizionalmente identifica il DIZIONARIO dei SINONIMI

quello che a noi interessa in realtà sono i TERMINI CORRELATI piuttosto che i SINONIMI.

Inizialmente i THESAURI erano composti solo da i termini con le loro liste di sinonimi

MODIFICA DELLA QUERY

Q BIANCO → BIANCO CANDIDO NIVEO (AUMENTO DEL RECALL)

↳ quando l'utente inserisce BIANCO nella query il sistema aggiungerà tutti i sinonimi presenti nel thesauro

Problema:

Non necessariamente dei termini sono sempre sinonimi (DIPENDE DAL CONCETTO)

Ad es. il dizionario calcistico è pieno di metafore belliche.
Il termine "CALCIO" inoltre ha più significati: avrà dunque dei sinonimi dal punto di vista ad es. del Gioco del calcio e altri sinonimi in CHIMICA.

Le tecniche utilizzate recentemente (1998) sono Thesauri organizzati come RETI SEMANTICHE

WORD NET è il 1° di questi Thesauri organizzati come reti semantiche (per l'inglese).

Viene usato massicciamente per fornire una serie di cose come:
- Modifica e arricchimento dell'interrogazione (AGGIUNTA DI TERMINI)
- Contestualizzazione dell'informazione
- Classificazione automatica dei testi (in cui, partendo da testi grezzi, si caratterizzano dal punto di vista semantico con una classificazione molto precisa. L'utente poi lavora su questa classificazione).

5. TERMINI RILEVANTI NON PRESENTI NEL DOCUMENTO

Questo problema può essere gestito, come visto precedentemente, con

WHITE LIST / KEY WORDS → Un classificatore umano attraverso l'uso di un vocabolario controllato, classifica il documento.

VANTAGGI: In teoria, il classificatore umano può classificare "accuratamente" il documento. → In pratica non avviene perché ad es. classificatori diversi usano termini diversi

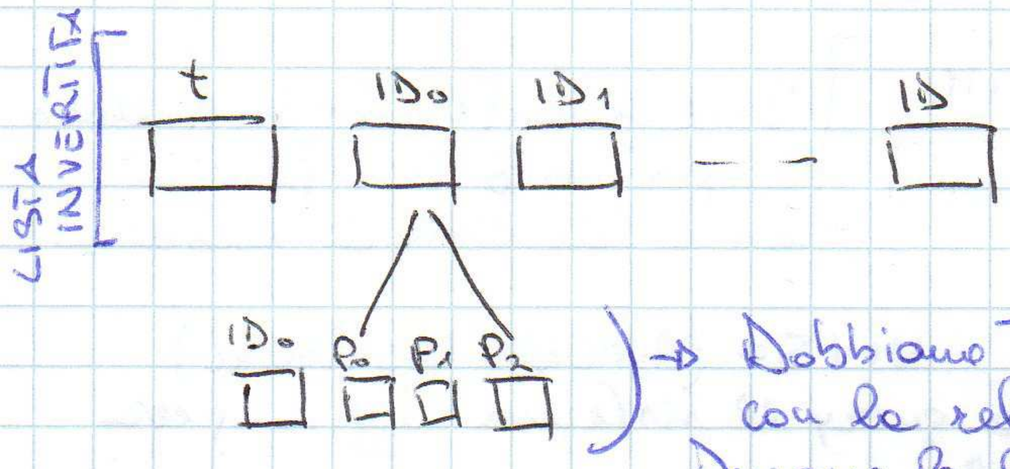
SVANTAGGI: Richiede lavoro umano (qualcuno deve leggere il documento e classificarlo);
Alcuni termini sono AMBIGUI. Ad es. il termine STRANIERI può rappresentare GIOCATORI STRANIERI e IMMIGRATI.

Se l'autore del doc. può dare le KEYWORDS, può imbrogliare.

A questo livello noi consideriamo i termini in qualunque posizione del documento
↳ Stiamo lavorando col modello BOOLEANO (ci dice se i termini compaiono o no nel doc).

Ad es. COMPUTER SCIENCE non è detto che nel documento compaiano vicino nel documento. → Con google per fare ricerca in questo modo usiamo mettere i termini fra virgolette
↳ Spesso invece siamo interessati alla ricerca di una frase

" COMPUTER SCIENCE "
↳ Viene richiesta la prossimità o l'adiacenza dei termini nel documento.



Per fare ciò le cose si complicano dal punto di vista degli indici. Nelle liste inv. descritte fino ad ora non abbiamo nulla che indichi la posizione del termine nel documento.
→ Dobbiamo tenere traccia di tutte le occ. di quel term. nel doc. con la rel. posizione per risolvere interrogazioni di questo tipo.
Dunque le liste invertite vengono processate facendo un AND dei termini e verificando che la posizione dei 2 termini sia adiacente o vicina.

I primi sistemi commerciali (IBM STAIRS) avevano un overhead delle strutture di indice che era 3 volte il CORPUS. Adesso si arriva ad avere strutture di indici con overhead del 30%...50% rispetto al corpus.

Il modello visto fino ad ora (MODELLO BOOLEANO) dà risp. ovviamente booleane (SI/NO) mettendo ~~tutti~~ allo stesso livello tutti i documenti considerati rilevanti → ciò non è vero

Noi vorremmo avere un RANKING DEL RISULTATO PER RILEVANZA DECRESCENTE

ROBERTSON & SPARCK-JONES hanno introdotto il

RELEVANT RANKING PRINCIPLE in cui si ORDINANO I DOC. NEL RISULTATO PER RILEVANZA DECRESCENTE
↳ I primi doc. della lista sono quelli più rilevanti

PROBLEMA: come calcolare la RILEVANZA di un documento rispetto alla query?

IPOTESI DI BLOOM: Si possono usare dei pesi per caratterizzare i termini
SCHEMI DI WEIGHTING: Servono a STIMARE (o approssimare) la RILEVANZA dei documenti rispetto all'interrogazione

Sono basati sulla FREQ. DEL TERMINE DEL DOCUMENTO.
Ovvero, se un utente chiede 'X' ⇒ Si assume che i doc. che contengono 'X' un # elevato di volte siano più rilevanti rispetto a quelli che lo contengono poche volte. Per giustificare questo tipo di approccio utilizziamo la tecnica della FUZZY LOGIC

A differenza della LOGICA BOOLEANA, secondo cui l'appartenenza di un elemento ad un insieme è caratterizzata da un funzione del tipo

$a \in A \begin{cases} SI \to 1 \\ NO \to 0 \end{cases}$, nella FUZZY LOGIC (→ LOGICA CHE MODELLA L'INCERTEZZA) l'appartenenza di un elemento ad un insieme è caratterizzata da una PROBABILITÀ

Questo è esattamente quello che succede nella nostra situazione, in quanto vogliamo verificare se un documento appartiene all'insieme dei doc. rilevanti e con che probabilità ci appartiene. Maggiore è la prob. di appart. e maggiore è la rilevanza del documento.

15a. Se usiamo brutalmente la freq. del termine nel doc., usando la tecnica della FUZZY LOGIC siamo in grado di stimare la RILEVANZA nel seguente modo:

FUZZY LOGIC → $e \in A$ con $P_e(A)$ | In questo modo artificialmente

QUERY	RILEVANZA
t	$f_t(d)$ → Se la query ha 1 termine la ril è la freq. del term. nel documento
$t \wedge t'$	$\min(f_t(d), f_{t'}(d))$ → minimo delle 2 probabilità (frequenze)
$t \vee t'$	$\max(f_t(d), f_{t'}(d))$ → max delle 2 freq
$t - t'$	$f_t(d) - f_{t'}(d)$ → Tolgo dalla fr. del 1° term. la fr. del 2°. Se ≤ 0 ⇒ ⇒ il doc. non è rilevante

Lo schema di WEIGHTING più comune è:

TF - IDF (TF = Term Frequency - IDF = Inverse Document Frequency)

Il TF più semplice è TF $f_{t,d}$ → freq. del term. t nel doc d (visto prima)

PROBLEMA 1: Non viene fatta alcuna normalizzazione sulla lunghezza del doc. La freq. dovrebbe essere normalizzata sulla lunghezza del doc.

PROBLEMA 2: Una stima di questo tipo favorisce i documenti in cui il termine compare più di frequente. → Secondo ZIPF termini molto freq. contano poco

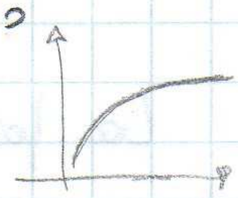
Dunque ciò che si usa in realtà è $tf_{t,d}$ (è più in accordo con la nostra percezione)

$$tf_{i,j} = \begin{cases} 1 + \underbrace{\log_2(f_{i,j})}_{\text{CORREZIONE LOGARITMICA}} & \text{se } f_{i,j} > 0 \\ 0 \end{cases}$$

i: termine, j: doc.

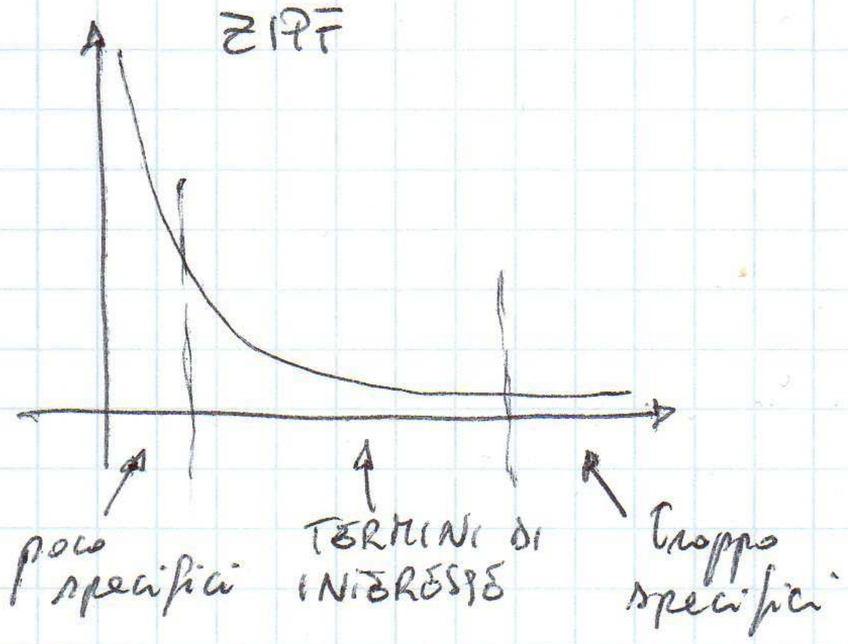
questo non considera la dimensione del documento

→ Man mano che aumenta la freq. ho una crescita meno rapida del $f_{t,d}$



IDF serve per calcolare l'importanza del termine all'interno del CORPUS.

L'idea è quella di MISURARE LA SPECIFICITÀ DEL TERMINE
↳ dal punto di vista statistico e non semantico



Con IDF si vuole "correggere" (pesare) la frequenza del termine con la specificità del termine, favorendo i termini più specifici tenendo conto del fatto che i termini estremamente frequenti comparirebbero in molti documenti e risultano dunque inutili

Vogliamo correggere questa frequenza perché termini di scarso interesse (come gli articoli il, lo, ...) potrebbero comparire con altissima frequenza.

Quindi vogliamo favorire i termini più specifici rispetto a quelli generali

Il TF-IDF non risolve comunque il problema ①

CARATTERIZZAZIONE DELLE FREQUENZE

156 Sia $n(r)$ = PIU' ALTA FREQUENZA AL LIVELLO DI RANKING r

$n(r) \sim r^{-\alpha}$ → costante empirica calcolata sulla base del corpus

FREQ. DOC. PER IL TERMINE k_i : $n(r) = C r^{-\alpha}$
↳ COSTANTE

→ Ragionevole per i testi in inglese

Se pongo $\alpha = 1$ e considero il logaritmo ottengo:

$$\log n(r) = \log C - \log r$$

Per $r = 1$ ottengo:
↳ Primo RANK

$$C = n(1) \rightarrow$$

In maniera alternativa a ciò si può usare $C = N$ per ottenere

$$\log r \sim \log N - \log n(r)$$

Possiamo definire la IDF in questo modo:

Sia k_i il Termine al rank r $\quad n(r) = n_i \Rightarrow idf_i = \log \frac{N}{n_i}$

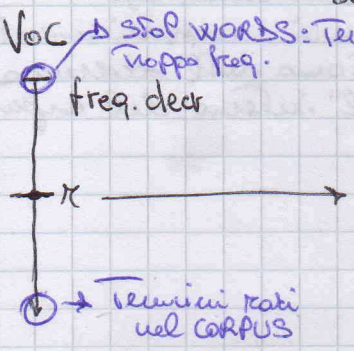
N → intero CORPUS ⇒ # di doc

n_i → freq di doc dello spec. Term.

La formula di peso diventa dunque:

$tf * idf$ → Peso per i vari Termini dell'interrogazione

Ordiniamo il VOC in maniera decr. per freq. dei doc. → VOGLIAMO CARATTERIZZARE LA CURVA CORRISPONDENTE

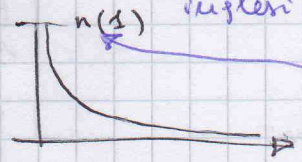


freq. doc per t di ORDINE r → # di doc. che sta all'r-esimo posto nell'ord.

$n(r) \sim r^{-\alpha} = \frac{1}{r^{\alpha}}$ → Serve per determinare quanto velocemente decade la curva

→ $\alpha = 1$ (Nel caso di CORPUS inglesi $\alpha = 1$)

ottenimo una curva del tipo $\frac{1}{x}$



Abbiamo bisogno di una NORMALIZZAZIONE che tenga conto della dim. del CORPUS

$n(r) = C r^{-\alpha}$ ⇒ Per $r = 1$ otteniamo $n(1) = C$ → Normalmente è circa N (vuol dire che il termine compare in tutti i documenti ad es. l'articolo 'il')

C → costante di normalizzazione; C → Dovrebbe essere il max # di doc selezionato da un termine

Passando al logaritmo stampo

$\lg n(r) = \lg C - \lg r$ Normalizzando (nel modo descritto prima) ottengo:

$\lg r \simeq \lg N - \lg n(r)$

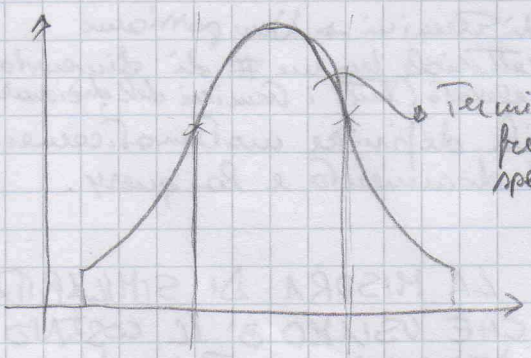
↳ Con C determiniamo il punto da cui parte la curva

Dunque possiamo definire la IDF nel seguente modo

$idf_i = \lg \frac{N}{n_i}$ → Varia da 0 a lg N. → per $n_i = 1$; ↳ per $n_i = N$

↳ idf del termine i

La formula di peso i data da $\boxed{tf * idf}$ → Dunque la freq. del termine viene corretta dalla componente idf che ha un andamento opposto a quello della freq. del termine tf



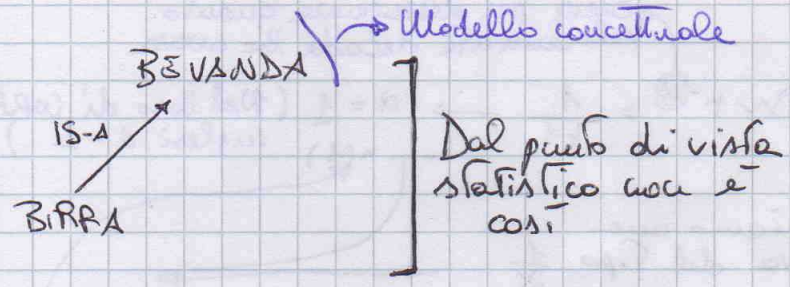
→ Termini sufficientemente frequenti con buona specificità → ZONA DI INTERESSE

Esistono tutta una serie di variazioni del peso come:

$TF\text{-}IDF = (1 + \lg_2 f_{i,j}) * \lg \frac{N}{n_i}$ → Normalmente si usa questo

$= f_{i,j} * \lg \frac{N}{n_i}$

Da notare che la specificità del documento prescinde dalle considerazioni concettuali. L'organizzazione concettuale dell'informazione non è necessariamente quella che ricaviamo dall'occorrenza dei termini all'interno del corpus.



Oltre al modello FUZZY, in precedenza era stato definito un altro modello:

MODELLO VETTORIALE (SALTON 60) → questo modello ha un'interpretazione geometrica/matematica molto precisa
↳ Viene utilizzato tutt'ora SMART

Definiamo uno SPAZIO VETTORIALE a T DIMENSIONI in cui OGNI DIMENSIONE CORRISPONDE AD UNO SPECIFICO TERMINE

I DOCUMENTI E LE QUERY VENGONO CARATTERIZZATI COME VETTORI

Documento: $D(w_0, \ldots, w_T)$ con w_i $\begin{cases} 0 & \text{se } i \notin D \\ \text{tf-idf} & \text{altrimenti} \end{cases}$

Query: $Q(w_0, \ldots, w_T)$

In questo caso la query è semplicemente una lista di termini → Non possiamo utilizzare operatori booleani. Altro prob: questo spazio vettoriale ha un # di dimensioni estremamente elevato (tutti i termini del dizionario)

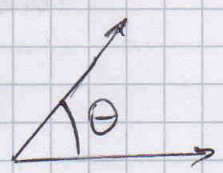
VANTAGGIO: con un'eq. di questo tipo è possibile definire matematicamente una misura di SIMILARITÀ tra il documento e la query.

$$S(x, y) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \, \|\vec{y}\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}}$$

prod. scalare

Norme calcolate prima

La SOMIGLIANZA è data dal cos dell'angolo tra i 2 vettori



LA MISURA DI SIMILARITÀ CHE USIAMO È IL COSENO DELL'ANGOLO TRA I DUE VETTORI $S(x,y)$

Il cos dell'angolo è 1 se l'angolo è 0° e 0 se l'angolo è 90°

Un altro coeff. di SOMIGLIANZA

<u>COEFFICIENTE DI JACCARD</u> (insiemistico - non è definito in uno spazio vettoriale)

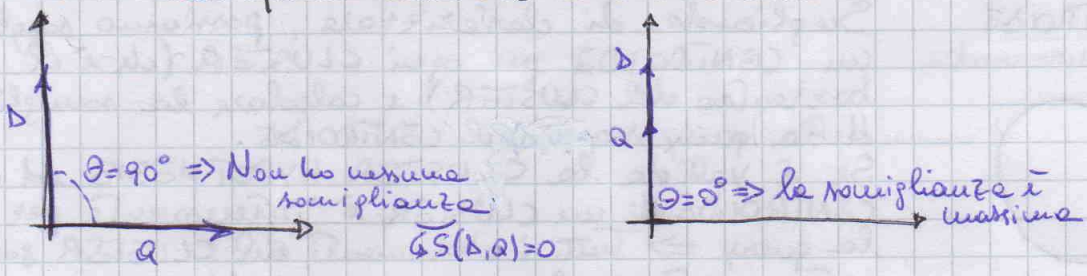
$$S(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

$|X \cap Y|$ → CARDINALITÀ DELL'INTERSEZIONE
$|X \cup Y|$ → CARDINALITÀ DELL'UNIONE

quando 2 insiemi sono uguali $S(X,Y)=1$ e quando 2 insiemi sono disgiunti allora $S(X,Y)=0$

↳ è più o meno la stessa cosa che avviene negli spazi vettoriali

Se ho uno spazio vettoriale a 2 dimensioni



Su questo modello (MOD. VETT.) abbiamo la stessa rappresentazione per i documenti e le query.
Questo ci permette di calcolare la somiglianza non solo tra documento e query ma anche tra documento e documento. → SI USA PER RAGGRUPPARE I DOC (CLUSTERING)
Su questo modello possiamo anche effettuare un raffinamento successivo della query aggiungendo dei termini

PROBLEMA: calcolo della funzione di somiglianza $S(X,Y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}}$
↳ calcolarla per ogni documento servono enormi risorse di calcolo

Tale problema può essere risolto col CLUSTERING: I documenti vengono raggruppati in gruppi omogenei in modo che il confronto non viene fatto più tra la query e tutti i doc. del CORPUS ma tra la query e un rappresentante di ogni <u>cluster</u>

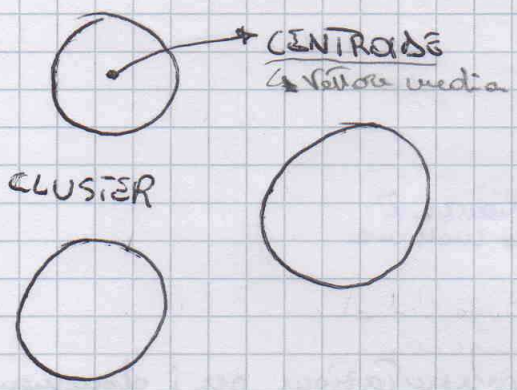
Un altro modo per risolvere tale problema deriva dal fatto che in realtà non dobbiamo considerare tutti i documenti del CORPUS in quanto per documenti disgiunti ho somiglianza = 0 (il doc. non viene recuperato).

Dunque possiamo semplicemente selezionare solo i documenti che non sono disgiunti con la query → eliminiamo i documenti di cui sappiamo a priori che producono 0.
Ovvero recuperiamo l'insieme dei documenti che contengono <u>almeno</u> un termine della query.
Per alleggerire ulteriormente il calcolo della funzione si ha che la norma di ogni documento viene fatto a priori (quando il doc. viene indicizzato)

CLUSTER HYPOTHESIS : DOCUMENTI SIMILI VENGONO REPERITI INSIEME

Questo vuol dire che se faccio un'interrogazione e recupero un doc ⇒ recupererò anche tutti i documenti simili a quello che ho recuperato.
Su questa base possiamo raggruppare i vettori dello SP. VETT. in CLUSTER in cui la SOMIGLIANZA dei documenti all'interno del gruppo sia maggiore tra i doc. del gruppo rispetto ai documenti degli altri CLUSTER.

Da ciò deriva che $1 - S(x,y)$ misura la distanza normalizzata tra i documenti.

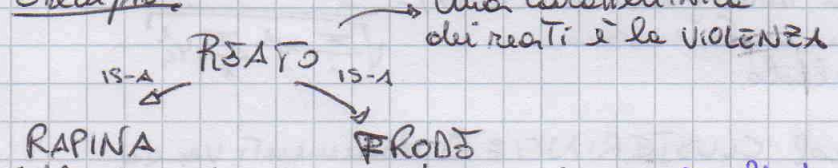


Scegliendo di clusterizzare, possiamo scegliere un CENTROIDE per ogni CLUSTER (che è il baricentro del CLUSTER) e calcolare la somiglianza della query rispetto al CENTROIDE.
Se è valida la CLUSTER HYPOTHESIS ed il CENTROIDE di un CLUSTER è "interessante per la query ⇒ tutti i documenti del CLUSTER sono interessanti per la query e quindi vengono recuperati anch'essi.

Questo semplifica di molto le cose in quanto il calcolo della somiglianza viene fatto solo sui CENTROIDI (e non su tutti i documenti)

Ovviamente, la SOMIGLIANZA È DI TIPO STATISTICO. ⇒ NON SI PUÒ ASSOCIARE UNA TEMATICA AI CLUSTER

Esempio:



RAPINA
↳ Nei doc. che parlano di rapina compare il termine violenza

FRODE
↳ Nei doc. che parlano di frode non compare il

⇒ I documenti relativi a FRODE e RAPINA possono appartenere a CLUSTER diversi (sono disgiunti) mentre l'utente dal punto di vista concettuale del REATO li considera congiunti.

La CLUSTERIZZAZIONE nasce in questo contesto ma viene utilizzata in tanti altri contesti (applicazioni di immagini, applicazioni audio, ...)

Dato che documenti e query vengono rappresentati in maniera uniforme (doc. e query sono entrambi vettori), possiamo usare direttamente i documenti come query.

MOD. AVANZATI DI DB — capitolo 8 del libro MODERN INFORMATION RETRIEVAL — 14/11/2012

MODELLING
RETRIEVAL EVALUATION
TEXT CLASSIFICATION
WEB SEARCHING

JAIN ET AL. DATA CLUSTERING: A REVIEW, ACM COMPUTING SURVEYS 31/3/1999

MECCANISMI DI CLUSTERIZZAZIONE:

Possiamo distinguere 2 tipi di CLUSTERING: PIATTO e GERARCHICO

CLUSTERING PIATTO: - K-MEANS

Si sceglie un numero k a priori.

1. SELEZIONIAMO K DOCUMENTI IN MANIERA CASUALE
   ↳ CENTROIDI dei CLUSTER
2. ASSEGNAMO OGNI DOCUMENTO AL CLUSTER CON IL CENTROIDE PIU' VICINO
3. AGGIORNIAMO I CENTROIDI
4. RIPETERE FINCHE' I CENTROIDI NON DIVENTANO STABILI

Quello appena descritto è l'algoritmo K-MEANS BATCH.
Esiste anche il K-MEANS ONLINE che ha una convergenza più rapida in quanto ricalcola i centroidi ad ogni assegnazione.

PROBLEMA: Scelta di k → è CRITICO.

PROBLEMA 2: La selezione casuale dei k documenti può portare a CLUSTER DIVERSI per SEMI diversi.
↳ IL SEME è la 1ª sel. di k doc.

CLUSTERING GERARCHICO:

BISECTING K-MEANS

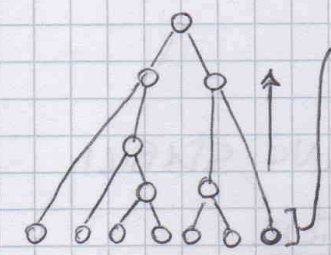
1. ASSEGNAMO TUTTI I DOC AD UN UNICO CLUSTER
2. PASSO DI SPLIT
   - SCEGLIAMO IL CLUSTER PIU' GRANDE
   - APPLICHIAMO K-MEANS CON k=2
3. RIPETIAMO FINCHE' IL CLUSTER PIU' GRANDE E' MAGGIORE DI UNA SOGLIA PREFISSATA

Con questo algoritmo fissiamo il # di documenti massimo che vogliamo avere all'interno di un CLUSTER (e non il # di CLUSTER come in K-MEANS BATCH e ONLINE)

CLUSTERING GERARCHICO

N = Numero di documenti

1. OGNI DOC. È UN CLUSTER
2. TROVIAMO I DUE CLUSTER PIÙ VICINI E LI FONDIAMO IN UN CLUSTER
3. RICALCOLIAMO LE DISTANZE TRA IL NUOVO E I VECCHI CLUSTER
4. RIPETIAMO 2 e 3 FINCHÈ NON OTTENIAMO UN UNICO CLUSTER



→ All'inizio ogni documento è un cluster.

Si fondono via via i documenti più vicini (si crea una gerarchia dal basso verso l'alto) finché non arriviamo ad avere un unico CLUSTER che rappresenta tutto l'albero. Ogni volta che fondiamo 2 CLUSTER abbiamo un FANOUT di 2.

PROBABILISTIC MODEL:

Ha portato alla definizione di una serie di stimatori di somiglianza: i BM

BM → BEST MATCH : Sono variazioni sul tema raffinati sperimentalmente

Il più interessante è il BM25:

Secondo il BM25 la SIMILARITÀ tra DOC. e QUERY è data da

$$SIM_{BM25}(d_j, q) \sim \sum_{k_i \in [q, d_j]} B_{ij} \times \lg_2\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

In cui n_i è il # di doc selezionati dal termine e in cui B_{ij} è:

$$B_{i,j} = \frac{(K_1+1) f_{i,j}}{K_1\left[(1-b) + b\,\frac{len(d_j)}{avg\ doc\ len}\right] + f_{i,j}}$$

i → Termine, j → doc.

$len(d_j)$ → lunghezza del doc j

$avg\ doc\ len$ → lunghezza media dei documenti

k_1 e b sono parametri empirici definiti di volta in volta. In genere si ha:

$k_1 = 1$, $b = 0.75$

Sperimentalmente questa formula (BM25) funziona meglio del VETTORIALE

19a Considerazioni su RECALL e PRECISION

Recall e Precision sono misure soggettive. Si presuppone dunque che qualcuno che conosce il CORPUS perfettamente, a fronte di una query sia in grado di determinare l'insieme di doc. rilevanti e quindi sia in grado di pesare RECALL e PRECISION in base ai doc. recuperati dal sistema. Ciò è possibile quando si ha a che fare con corpora piccoli. Risulta invece impossibile calcolare RECALL e PRECISION per corpora molto grandi (come il WEB)

Come già visto, il problema del RECALL è principalmente un problema di GAP SEMANTICO tra il modello del sistema e quello dell'utente.

Se vogliamo confrontare la qualità di implementazioni diverse basate sulle stesse definizioni di termini, quello che ci interessa maggiormente è la PRECISION in quanto su corpora molto grandi è utopistico pensare che l'utente vada a vedere tutti i documenti rilevanti.
Dunque la PERDITA DI DOCUMENTI in corpora come il WEB è accettabile mentre il RUMORE soprattutto nei primi documenti è decisamente meno accettabile.

Dunque ciò che generalmente viene utilizzato per confrontare sistemi diversi è la PRECISION a 2 punti precisi:

- PRECISION @ 5 → Precisione complessiva dei primi 5 documenti
- PRECISION @ 10 → Precisione complessiva dei primi 10 documenti

Questo tipo di valutazione permette di effettuare valutazioni soggettive ad utenti che non conoscono l'intero corpora. È sufficiente dunque che il valutatore (utente) scorra i primi 5 o 10 documenti e verifichi quanto sono precisi rispetto all'interrogazione. → In pratica stiamo valutando l'accuratezza del RANKING