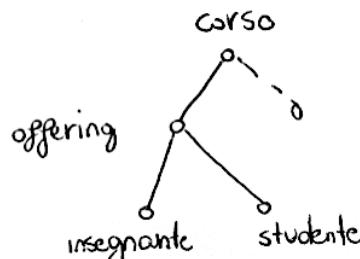


Modelli di DB

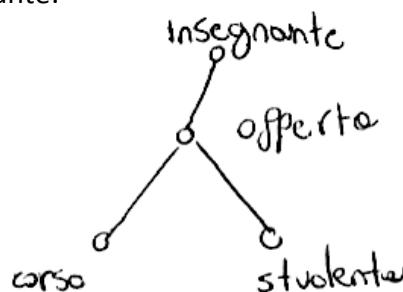
- cronologia:
 - ▶ modello gerarchico (anni 60)
 - ▶ modello reticolare (anni 70)
 - ▶ modello relazionale (anni 80)
 - ▶ modello a oggetti (anni 90)

Modelli gerarchico

- i dati sono rappresentati come record
- le associazioni tra i dati sono rappresentate con puntatori in una struttura ad albero (grafo non orientato e aciclico)
- abbiamo quindi una collezione di record collegati da archi che rappresentano le associazioni
- negli anni 60 si utilizzavano i nastri magnetici --> accesso sequenziale
- da un nodo padre si riesce a raggiungere il nodo figlio ma non viceversa
- la caratteristica del modello gerarchico è che ogni nodo ammette un solo genitore
- es:



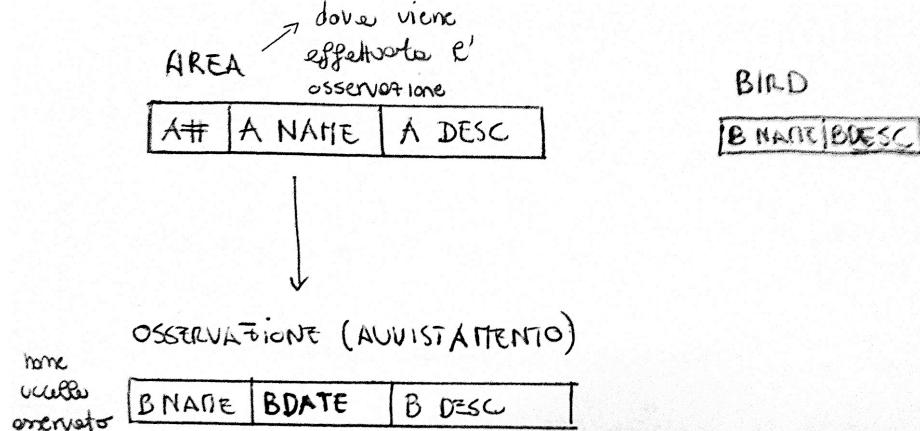
- vantaggi: linearizzazione dell'albero (indispensabile con i nastri magnetici)
- svantaggi:
 - ▶ le relazioni sono solo 1-N
 - ▶ asimmetria:
 - ho un ordine di accesso predefinito
 - per effettuare delle ricerche devo attraversare tutto l'albero
 - ho delle query risolte bene e alcune male (es ricerca degli insegnanti)
 - ▶ anomalia:
 - non può esserci uno studente che non è iscritto a un corso
- evoluzione 1 (per risolvere asimmetria)
 - ▶ la soluzione è rappresentata dalla duplicazione dei dati
 - ▶ ovvero si costruisce una nuova gerarchia con diversa radice
 - otteniamo gerarchie con diversa radice
 - ▶ se voglio partire da insegnante:



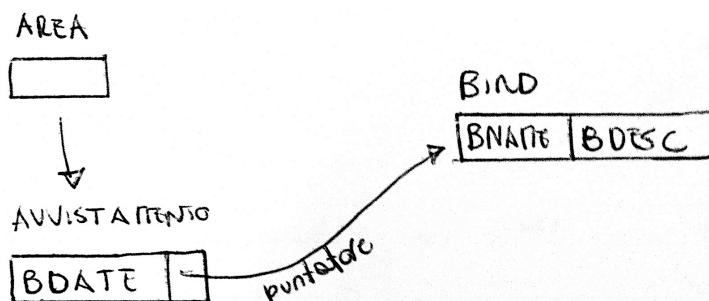
- ▶ svantaggi: abbiamo una ridondanza dei dati e quindi elevati costi di storage

- evoluzione 2 (per risolvere ridondanza)

- ▶ per risolvere questo problema si introduce un meccanismo a puntatori
- ▶ invece di duplicare i dati vengono create più gerarchie collegate tra di loro tramite puntatori
- ▶ es: consideriamo un DB di osservazioni ornitologiche

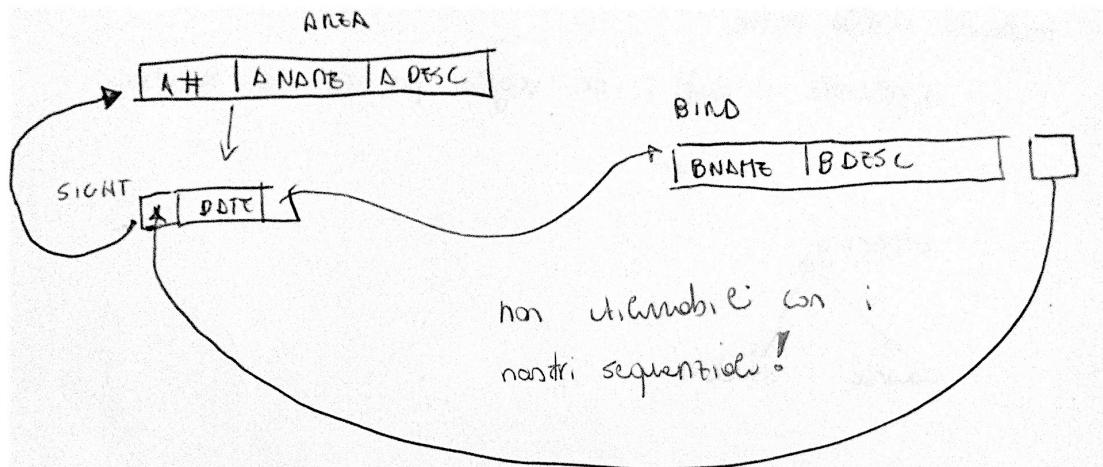


- c'è una duplicazione del BNAME e BDESC che vengono riferiti al osservazione: per evitare questa duplicazione eliminiamo i dati da osservazione e usiamo un puntatore per puntare ad uccello



- ▶ su una stessa area possiamo avere più osservazioni
- ▶ è facile partire dall'area, è molto difficile partire dall'osservazione
- ▶ se vogliamo partire da *avvistamento* e non dall'*area* invece di duplicare con una nuova radice (avvistamento--> area), tolgo i dati BNAME e BDESC dall'*osservazione*, lasciando la *data* e aggiungendo un puntatore che punta alla descrizione dell'uccello
- ▶ questo permette di arrivare in qualche modo dall'*area* all'*uccello* ma non ci permette di navigare la struttura in modo libero (devo partire sempre da area e poi finiremo alla descrizione dell'uccello)

- evoluzione 3 (per risolvere problema della navigazione)
 - ▶ stabilisce delle associazioni logiche bidirezionali
 - ▶ mediante i puntatori simulo delle gerarchie di accesso senza duplicare i dati
 - ▶ es:

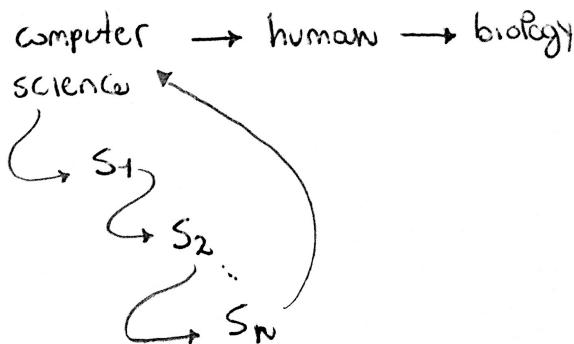


- ▶ vantaggi: rende tutto navigabile in modo arbitrario (posso partire da dove voglio)
- ▶ svantaggi:
 - diventa più complicato
 - troppi puntatori
 - non utilizzabile con i nastri ma solo con i dischi ad accesso diretto
- ▶ vedendo la cosa come relazioni si nota che partiamo definendo una relazione e definiamo poi esplicitamente l'inverso (questo non succede nel modello relazionale)

Modello reticolare

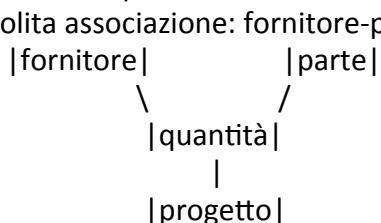
- i dati sono rappresentati come record
- le associazioni tra i dati sono rappresentate con puntatori in una struttura a grafo
- è basato pesantemente su puntatori
- abbiamo dei tipi di record
- possiamo anche avere relazioni M-N (partiamo comunque da 1-N)
- es

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>Dipartimento</td></tr> <tr><td style="height: 20px;"></td></tr> <tr><td>studente</td></tr> </table>	Dipartimento		studente	owner (padre) set (associazione tra owner e member) member (figlio)
Dipartimento				
studente				



- rappresentazione fisica e enumerazione dei vari dipartimenti (dipartimenti: computer science, human, biology)
 - ▶ gli studenti sono collegati tra di loro tramite puntatori e l'ultimo studente è riconnugliato al padre
 - se vogliamo enumerare tutti gli studenti partiamo dal 1° dipartimento scendiamo e li enumeriamo, passiamo al 2° dip ecc. ecc.
 - per risolvere query attraversiamo la struttura di puntatori

- permette multimember sets: i membri di una relazione possono essere di tipo diverso
- possibilità di avere sets recursivi: impiegato e supervisore (il supervisore è un impiegato)
- permette sets speciali: non hanno l'owner (l'owner è il sistema)
- com rappresentiamo M-N (non è banale):
 - ▶ supponiamo di avere la solita associazione: fornitore-parte-progetto



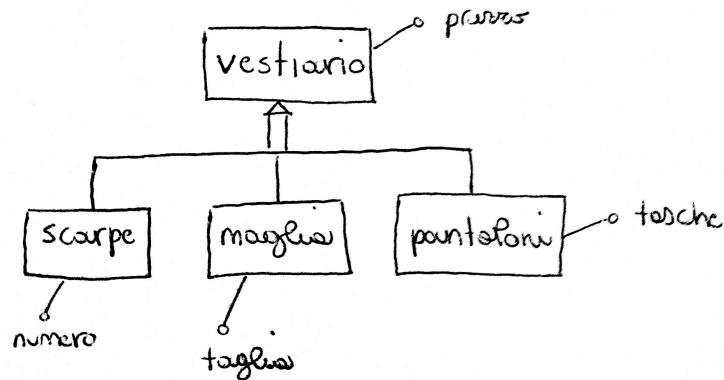
- ▶ soluzione: linking o dummy record type
- ▶ fornitore, parte e progetto diventano l'owner del linking quantità
- ▶ si utilizza un record type fittizio (quantità) che funziona da member (figlio) di più owner (padre)
- ▶ svantaggi:
 - interazione/navigazione completamente procedurale: il programmatore segue i link (pesante per il programmatore)
 - non esiste separazione tra livello logico e fisico
 - il programma ha visibilità logica e fisica di tutto: es se nel progetto voglio passare da B+-tree a hashing devo cambiare programma

Modello relazionale (codd 1970)

- con la nascita del modello relazione le cose si semplificano
- vantaggi:
 - ▶ abbiamo una separazione (indipendenza) tra la parte logica e la parte fisica (non così marcata a causa del concetto di record)
 - schema logico e schema fisico
 - puntatori logici e puntatori fisici
 - ▶ Codd definisce un modo matematico rigoroso (algebra relazionale !!) non procedurale per definire il risultato di un interrogazione
- svantaggi:
 - ▶ se il DB era reticolare dovevo riscrivere tutto
 - ▶ i primi sistemi relazionali erano molto lenti
- recuperiamo insiemi di dati anzi che un dato alla volta
- semplice (dal punto di vista concettuale e della manipolazione) ma completo
- caratteristiche (Internet)
 - ▶ il DB viene rappresentato come un insieme di tabelle
 - ▶ viene considerato attualmente il modello più semplice ed efficace, perché è più vicino al modo consueto di pensare i dati, e si adatta in modo naturale alla classificazione e alla strutturazione dei dati
 - ▶ le associazioni tra i dati sono ottenute associando valori di attributi in tabelle diverse (join?)
- le limitazioni di questo modello emergono abbastanza presto: **critica di Kent** (di tutti i modelli basati su record: gerarchico, reticolare e relazionale)
 - ▶ record: "è un sequenza fissa di campi conformi a una descrizione statica (fatta a priori)"
 - ▶ critica:
 - anomalia orizzontale
 - anomalia verticale
- considerazione record-base: viene utilizzata l'aggregazione (record) come strumento di modellazione
 - ▶ questa è una considerazione di tipo fisico che può avere impatto negativo: se voglio sapere solo il *nome* di *persona* accedo comunque a tutto il record!

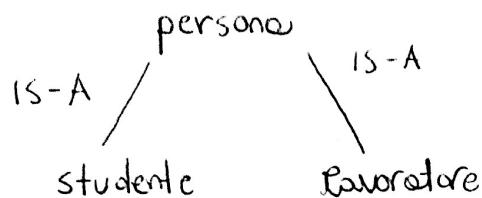
- **anomalia/disomogeneità orizzontale:** (record visto orizzontalmente)

- ▶ i fatti rilevanti sono tutti omogenei: ossia quello che noi vogliamo registrare su un entità è sempre composto dagli stessi attributi (è statico)
- ▶ Kent sostiene che questa assunzione non è necessariamente vera
 - es vestiario: ci sono attributi che cambiano da vestito a vestito
- ▶ porta come conseguenza al meccanismo di generalizzazione/specializzazione della gerarchia di tipi (soluzione di kent?)
- ▶ le disomogeneità orizzontali derivano dal fatto che appiattiamo una gerarchia di tipi (concettuali) considerandola come un unico record
 - Kent fa presente che esiste una gerarchia di tipi!:



- altro es:

se appiattiamo la struttura e considero in una persona i campi di lavoratore questi saranno a NULL!



- **anomalia/disomogeneità verticale:** (record visto verticalmente)

- ▶ in un tipo di record, un campo contiene sempre lo stesso tipo di valori (non va bene)
 - il problema è che molte volte vogliamo usare un unico campo per rappresentare tipi diversi
- ▶ es: consideriamo il campo azionista:
 - può essere: persona fisica o persona giuridica
 - possono essere identificati in modo diverso: P.IVA e CF!

- conseguenze delle critiche di ket (per superare questi due problemi??):
 - ▶ 1. reticolo semantico -> gerarchia di tipi
 - per risolvere la disomogeneità orizzontale
 - ▶ 2. identificazione delle tuple / istanze(di record)
 - per risolvere la disomogeneità verticale
 - **1. reticolo semantico -> gerarchia di tipo** (gia visto)
 - ▶ risolviamo la disomogeneità orizzontale definendo dei reticoli semantici (ovvero una gerarchia di tipo)
 - ▶ gerarchia di tipo => ereditarietà
- ```

graph TD
 persona -- CF --> CF
 persona -- name --> name
 persona -- matricola --> matricola
 persona --- studente
 persona --- lavoratore
 studente --- matricola
 lavoratore --- società
 lavoratore --- stipendio
 studente -.-> lavoratore
 studente - lavoratore --- matricola
 studente - lavoratore --- società
 studente - lavoratore --- stipendio

```
- ▶ **a. ereditarietà (multipla?) dei dati**
    - ci sono i noti problemi di ambiguità dei dati
  - ▶ **b. proprietà di inclusione delle istanze (sussunzione)**
    - la relazione di **sussunzione** spesso è chiamata IS-A (sottoclasse/sottotipo)
    - in pratica imponiamo dei vincoli
    - $A \leq B$  implica che  $\text{ist}(A) \subseteq \text{ist}(B)$ 
      - $A \leq B = B$  sussume A
      - $\text{studente} \leq \text{persona} \rightarrow \text{ist}(\text{studente}) \subseteq \text{ist}(\text{persona})$ 
        - ▶ persona sussume studente
    - ▶ in pratica quando stiamo definendo una gerarchia di tipi stiamo definendo dei vincoli sulle istanze (molto spesso abbiamo una relazione di IS-A)
      - un modello in grado di rispondere alla critica di Kent deve supportare dei reticoli semantici
      - quello che si vuole supportare è il concetto di ereditarietà multipla, la quale 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 due oggetti della gerarchia e definire un nuovo oggetto (es: studente lavoratore!)
    - ▶ se appiattiamo trattiamo tutti come studenti lavoratori: *CF, nome, matricola, società, stipendio* --> se mi interessa persona uso solo *CF, nome* mentre gli altri campi sono a NULL
      - questo perchè abbiamo appiattito una cosa che non è piatta
      - complichiamo la cosa perchè molti campi verranno messi a NULL
      - il discorso del reticolo semantico è un discorso che diventa naturale per descrivere situazioni in cui in realtà le entità che stiamo descrivendo sono diverse ma sono collegate tra di loro da relazioni da IS-A (tipicamente)
        - studente IS-A persona vuol dire che studente eredita le proprietà di persone e eventualmente ne aggiunge di altre
    - ▶ dalla sussunzione (=imporre dei vincoli) derivano i concetti di estensione superficiale e profonda

- dobbiamo considerare le entità (persona, studente, lavoratore) in maniera separata: per esempio le persone sono solo persone (ne studenti ne lavoratori), senza i dati dello studente/lavoratore
- **estensione superficiale:** considero la persona in generale senza considerare il fatto che sia studente o lavoratore (senza i dati dello studente/lavoratore)
- **estensione profonda:** il set di istanze di persone è:  
est\_sup di persona *unione* est\_sup studente *unione* est-sup lavoratore

## • 2. identificazione delle tuple / istanze(di record)

- siamo nel contesto della disomogeneità verticale
- il problema che pone Kent è principalmente nell'identificazione delle istanze
- il modo con cui identifichiamo il record nel modello relazionale è la chiave primaria (PK) la quale deve essere unica all'interno della relazione
- caratteristiche PK
  - **unicità**
  - **persistenza**
- problemi che si possono verificare:
  - a. non uniformità della PK
    - la chiave primaria delle persone giuridiche è diversa da quella delle persone fisiche (P.IVA vs CF)
  - b. non persistenza
    - se una persona è identificata tramite CF e questo cambia ?
- soluzione: utilizzare un ID univoco che non dipende da CF
  - utilizzo un surrogato: un ID opaco definito dal sistema
  - è **persistente** nel senso che non varia per tutta la vita dell'entità
  - risolviamo anche il problema della **non uniformità**: identifico tutti allo stesso modo
  - CF non è più una PK ma diventa una **chiave candidata**: unico vincolo è la unicità

- la gerarchia di tipi e l'identificazione tramite surrogati li abbiamo sia in OO che OR
- ne l'OO ne l'OR risolvono completamente le critiche di Kent:
  - ▶ il problema è di abitudine mentale: quando parliamo di record (una sequenza fisica di campi) quello che abbiamo in mente è in realtà una memorizzazione contigua di questi dati
  - ▶ usare un record come strumento di memorizzazione contigua dei campi del record, non è, sotto certi punti di vista, necessario né vantaggioso
  - ▶ il record è un aggregazione fisica: sto portando degli aspetti fisici (memorizzo in maniera contigua) a livello logico
- ▶ **problema 1:**
  - usando l'aggregazione abbiamo dei problemi semantici: l'aggregazione modella la relazione parto-off:
  - ma gli attributi molto spesso non sono parte dell'entità: l'età di una persona non è una parte di una persona... è una associazione ma non un aggregazione!
  - mentre il motore è una parte dell'automobile....
  - quindi l'aggregazione implica la relazione di parto-of!
- ▶ **problema 2** (può essere molto più grave del precedente):
  - quando passiamo da EA --> relazionale, la prima regola dice: per ogni entità creiamo una relazione dove mettiamo tutti gli attributi
  - questo vuol dire che abbiamo una relazione molto molto grande: dal punto di vista fisico questo non è desiderabile: se voglio accedere al nome di uno studente tiro su anche tutte gli altri campi che compongono il record studente!
- ▶ il concetto di record come aggregazione fisica di campi non serve a livello logico e può essere dannoso a livello fisico! ma tuttavia continua a persistere...
- ▶ **problema 3:**
  - le strutture sono instabili
  - se ho dei cambiamenti nelle descrizione dei dati (es. per legge devo aggiungere l'altezza dello studente), devo per forza effettuare delle modifiche fisiche!
- ▶ in memoria centrale questo concetto di record non da problemi, ma in memoria secondaria può essere molto influente

## Evoluzione

- 1. intorno agli anni 60/70 nasce il concetto di **abstract data type**:
  - ▶ idea di **incapsulamento**: scatola nera di cui conosciamo input e output ma non la rappresentazione interna
  - ▶ c'è un qualcosa che ha delle **variabili interne** non accessibili direttamente ma solo tramite **appositi metodi**
    - questo ci permette di cambiare la struttura interna della scatola nera senza che i programmi esterni subiscano variazioni
  - ▶ (wikipedia)
    - un ADT è un tipo di dato le cui istanze possono essere manipolate con modalità che dipendono esclusivamente dalla semantica del dato e non dalla sua implementazione
    - nei linguaggi di programmazione che consentono la programmazione per ADT, un tipo di dati viene definito distinguendo nettamente tra
      - la sua interfaccia: ovvero le operazioni che vengono fornite per la manipolazione del dato,
      - la sua implementazione interna: ovvero il modo in cui le informazioni di stato sono conservate e in cui le operazioni manipolano tali informazioni al fine di esibire, all'interfaccia, il comportamento desiderato
    - la conseguente inaccessibilità dell'implementazione viene spesso identificata con l'espressione **incapsulamento** (detto anche **information hiding**: nascondere informazioni).
  - ▶ (internet)
    - I tipi di dati astratti stanno alla base dell'incapsulamento, poiché permettono di nascondere i dettagli dell'implementazione e di favorire il riutilizzo
    - esempi di ADT sono le liste con priorità, le quali possono essere implementate in vari modi, utilizzando anche altri ADT
- 2. intorno al 70 smaltalk introduce l'idea di **gerarchia di tipi** e quindi l'**ereditarietà dei dati e delle operazioni**
  - ▶ prima gli abstract data type erano piatti! -> no gerarchia?
  - ▶ la gerarchia dei tipi semplifica il **riuso** e il **disegno incrementale** in quanto permette il disegno per raffinamenti successivi
    - quindi ho ho gerarchie di astrazione in cui si va specializzando il tipo, facendogli fare un qualcosa in più rispetto al padre)
  - ▶ insieme all'ereditarietà si ha il **polimorfismo o overloading**
    - possiamo definire la stessa operazione con lo stesso nome su oggetti diversi (un metodo che fa la stessa cosa su oggetti diversi e ha lo stesso nome)
      - vengono utilizzati metodi diversi a seconda dell'oggetto
    - questo provoca ancora una semplificazione del disegno
- queste due motivazioni (1 e 2) messe insieme portano alla nascita dei sistemi Object-Oriented (OO)
  - ▶ OO è una disciplina di disegno particolare
  - ▶ OO non ha avuto poi così successo nel campo dei DB

## Motivazioni di ingegneria del software (per la nascita dei DBMS OO??)

### • impedance mismatch

- ▶ la separazione tradizionale tra linguaggi di programmazione e DB dovrebbe essere evitata (ambienti diversi che cerchiamo di mettere assieme)
- ▶ **idea OO:** usare un unico ambiente con **dati persistenti**
  - ossia eliminiamo la parte embedded (incorporata) sul DB e consideriamo il DB come avente dei dati (come nei linguaggi di programmazione) con la differenza che i dati sono persistenti

### • applicazioni con oggetti complessi

- ▶ CAD, CAM, sistemi multimediali, motori di ricerca
- ▶ abbiamo oggetti non tradizionali (testi, video, suoni) che non sono gestiti direttamente dal sistema di DB
  - vogliamo aumentare le capacità del sistema per gestire oggetti che non sono abstract data type

### • performance

- ▶ i Db relazionali sono lenti (a quei tempi), ma il “programmatore astuto” se usa un ambiente unico con dati persistenti, può programmare l’accesso ai dati in maniera efficiente (usato come strumento di marketing)

### • alcuni problemi di questo approccio uniforme (??) (OO) sono:

- ▶ l’incapsulamento
  - per l’ingegneria del sw è un vantaggio!
  - se l’accesso ai dati può avvenire solo tramite apposite funzioni, impedisce al programmatore di usare tutta una serie di cose (ad es. indici)
  - non usato in ambiente DB
- ▶ associazioni
  - l’approccio OO puro lavora su un modello funzionale in cui le associazioni vengono rappresentate come attributi di oggetti (analogamente alla rappresentazione che usiamo dei DB relazionali)
  - |EMP| -|----->|DEPT|
    - all’interno dell’oggetto EMP ho un riferimento (puntatore) che punta all’oggetto DEPT
    - per arrivare a DEPT partendo da EMP devo definire anche il puntatore inverso!
      - ▶ torno ad avere troppi puntatori come nel modello gerarchico e reticolare
  - nel modello relazionale (è di tipo tabellare) questo viene fatto in maniera molto più agevole
  - quindi il passaggio al modello funzionale (OO) complica ulteriormente le cose:
    - perchè devo sempre definire l’inversa (con funzioni che hanno molti argomenti diventa complicato)
    - perchè sparpagliamo i dati mantenendo sempre un record ma perdendo quello che è l’insieme dei record (ovvero la tabella)

## OODBMS: standard ODMG 2.0

- (internet)

► tipo:

- tipo di dato astratto o classe
- definendo una classe si crea un nuovo tipo di dato

► oggetto:

- sito 1: istanza di un tipo
- sito 2: Il termine istanza è quasi simile al termine oggetto; se ne differenzia in quanto sottolinea l'appartenenza dell'oggetto a un dato tipo (istanza di ... "qualcosa")

- esempio, la dichiarazione/definizione:

*int ivar;*

► costruisce l'oggetto ivar, istanza del tipo int.

- sito 3: In realtà si dice che un oggetto è una instanziazione di una classe, ragione per cui si può parlare indifferentemente di oggetto o di istanza (eventualmente di occorrenza)
- sito 4: un oggetto è un'istanza di una classe

- concetto di **identità dell'oggetto**:

- è un puntatore
- non è solitamente un indirizzo fisico
- identificativo unico e immutabile
- indipendente dalle altre caratteristiche dell'oggetto (no CF come identificativo)
- l'id dell'oggetto (OID) viene mappato in un indirizzo fisico tramite una tavola di traduzione (solitamente tavola hash)
- **pointer swizzling** (scambio di puntatori) --> è una tecnica
  - una volta che utilizziamo l'indirizzo fisico dell'oggetto, sostituiamo o copiamo l'indirizzo fisico in quello logico
  - da quel momento non abbiamo più bisogno della tavola di traduzione, ma accediamo direttamente all'indirizzo fisico (in memoria primaria)
  - | \* | OID | indirizzo fisico |
    - \* è un qualcosa che dice: "è un OID!"

- c'è una suddivisione tra **oggetti** e **valori letterali**

► **1. oggetti** (che sono tipicamente **complessi**)

- tipi di oggetti complessi
  - **strutturati**: quelli classici
  - **non strutturati**: sono spazzi 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 (es: immagini, video, testi...)
- sono identificati tramite OID

► **2. valori letterali**:

- non hanno OID
- hanno solo il valore letterale

- gli oggetti sono costruiti da altri oggetti o valori, mediante l'applicazione dei **costruttori di tipi**

- i **tipi** che utilizziamo per costruire gli oggetti sono predefiniti

► **ATOM**: oggetto atomico

► **collezioni** (sono raggruppamenti di oggetti / sono collezioni di OID)

- **SET**: insieme
- **LIST**: insieme ordinato
- **BAG**: multinsieme
- **ARRAY**: vettore

► **TUPLA**:

- costruttore di tipo strutturato (struct?? )

- è un aggregato di OID (come una tupla del sistema relazionale)

- vediamo come costruire un oggetto con queste tecniche:

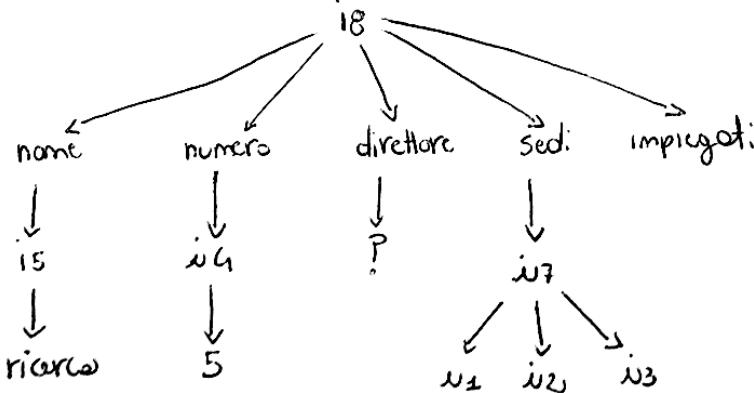
- ▶ identifico un oggetto come una terna: OID, costruttore, valore

- es

- O1 (i1, ATOM, "huston")  
 O2 (i2, ATOM, "bellaire")  
 O3 (i3, ATOM, "sugarlanda")  
 O4 (i4, ATOM, "5")  
 O5 (i5, ATOM, "ricerca")  
 O6 (i6, ATOM, "1988-05-22")

- ▶ aggreghiamo questi valori per definire nuovi oggetti

- O7 (i7, SET, {i1, i2, i3}) (O7 sedi di un dipartimento)
- O8 (i8, TUPLA, <nome\_dipartimento: i5, numero\_dipartimento: i4> direttore\_dipartimento: i9, sedi\_dipartimento: i10>)
- viene definita implicitamente una struttura a grafo che parte da O8:



- con l'applicazione dei vari tipi di costruttori abbiamo definito una struttura a grafo, in cui possiamo trovare i valori navigando il grafo
- come detto, i costruttori predefiniti costruiscono un oggetto mettendo assieme *valori atomici, collezioni o tuple*

- non sono gli oggetti classici di java!

- in questo contesto diventa difficile il concetto di **uguaglianza tra oggetti**

- ▶ dati 2 oggetti *ik* e *ij* posso avere due tipi di uguaglianza

- **uguaglianza profonda**

- stessa struttura
- stessi valori (stesso stato)
- stesso OID
- ovvero: grafi esattamente uguali

- **uguaglianza superficiale**

- rilassa il concetto di uguaglianza precedente concentrandosi sullo stato
  - ▶ è meno restrittiva della precedente
- stessa struttura
- stessi valori (stesso stato)
- OID possono diversi

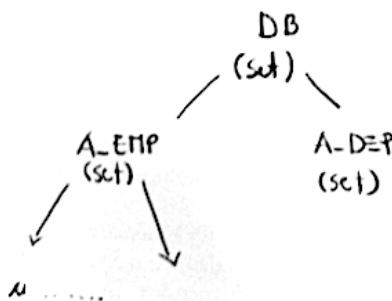
- gli OID usati da altri oggetti rappresentano delle associazioni (relazioni?) tra oggetti diversi
  - ▶ l'**associazione binaria** (associazione tra due oggetti) può possedere un riferimento inverso (che viene definito esplicitamente)
  - ▶ es
    - O1(i1, TUPLA, <nome\_impiegato: i2, dip\_impiegato: i3,...>)
    - O2(i2, ATOM, "Rossi" )
    - 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 effettivamente un associazione binaria siamo costretti a definire appositamente una funzione inverso (esattamente come nel modello reticolare con la differenza che qui il livello fisco rimane "del tutto" trasparente)
- **incapsulamento** (information hiding)
  - ▶ con l'abstract data type abbiamo un encapsulamento totale: i dati sono invisibili esternamente e sono accessibili solo attraverso le operazioni definite all'interno dell'ADT
  - ▶ dal punto di vista dei DB questo è molto 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 rilassato: ci sono attributi in parte visibili e in parte nascosti
    - **stato --> proprietà/attributi**: tipicamente sono visibili in lettura e gestite scrittura dai metodi
      - **visibili**
      - **nascosti**
    - **operazioni** (se abbiamo solo la dichiarazione abbiamo una classe **astratta** e non è istanziabile)
      - **interfaccia/singature**: prototipo della funzione
        - ▶ interface:
          - definizione di tipi astratta
          - definiamo le operazioni che il tipo esporta ma l'interface non è direttamente istanziabile
        - **metodo/implementazione**: implementazione della dichiarazione della funzione
  - ▶ l'aggiornamento solitamente viene fatto in maniera encapsulata perché permette di gestire in maniera opportuna i vincoli di un specifico oggetto
  - ▶ il modo per definire l'incapsulamento è la definizione di classe
    - classe: definizione di
      - **tipo**
      - **operazioni**
  - ▶ si usa la **dot notation** per indirizzare variabili o operazioni di uno specifico oggetto
    - questa notazione può essere estesa: **path expression**

*impiegato < dipartimento , nome >*      *impiegato*  
*↳ dipartimento < nome , direttore >*

- posso comporre le cose scrivendo: *e.dipartimento.direttore.nome*
  - ▶ dove *e* è un'istanza di *impiegato*
- in termini relazionali stiamo definendo dei join in maniera molto più naturale rispetto a quando lo facciamo in termini relazionali

- **persistenza**

- ▶ non tutti gli oggetti sono persistenti
  - **oggetti temporanei** (servono per la logica del programma: muoiono con il processo)
  - **oggetti persistenti** (memorizzati sul DB)
- ▶ metodi per rendere persistente un oggetto
  - 1. per **nome**
    - diamo un nome agli oggetti
    - il DB ricorda questo nome e attraverso di esso mi permette di accedere all'oggetto
    - possiamo dare un nome a un numero limitato di oggetti
  - 2. per **raggiungibilità**
    - B è raggiungibile da A se esiste una sequenza di riferimenti che da A porta a B (il *dipartimento* è raggiungibile da *impiegato*)
    - se A è persistente anche ogni B raggiungibile da A è persistente
    - se combiniamo i due tipi di persistenza



- ▶ possiamo definire un oggetto persistente con un nome (es il nome che do al DB)
- ▶ tutto ciò che è raggiungibile dall'oggetto 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*, *dipartimento* rendendoli persistenti
  - tutto ciò che è raggiungibile da loro diventa persistente
  - i nomi che usiamo in questo modo sono in numero limitato e sono di ugual numero rispetto a un DB relazionale
  - stiamo simulando delle relazioni: *A\_EMP* corrisponde (nel modello relazionale) alla relazione che contiene tutti gli impiegati
- ▶ questo meccanismo che permette di raggruppare logicamente gli oggetti viene chiamato **extent** e fa parte dello standard ODMG 2.0

- **gerarchia di tipi con ereditarietà:** già visto precedentemente

- **oggetti complessi**

- ▶ differenze tra oggetti/valori letterali

- oggetti
  - hanno OID
  - hanno vita propria
  - reference semantic
- valore letterale
  - es: nomi, stipendio...
  - è associato ad altri oggetti
  - non hanno OID
  - ownership semantic
- NB: eliminano OID dagli oggetti che non ne hanno bisogno

- **ownership semantic**

- i valori letterali sono incapsulati all'interno dell'oggetto padre e dipendono da lui (se cancello l'oggetto padre cancello anche i valori)

- **reference semantic**

- componente oggetto indipendente, riferito tramite il suo OID

- es

(i1, TUPLA, <nome: i2, dipartimento: i3>)

(i2, ATOM, "Rossi")

(i3, TUPLA, <nome\_dipartimento: i4, ... >)

- i2 non ha senso che viva per conto suo

- è inefficiente! posso rappresentare "Rossi" come un valore letterale, facendolo diventare una componente di un altro oggetto

- ottengo: (molto più efficiente)

(i1, TUPLA, <nome: "Rossi", dipartimento: i3>)

(i3, TUPLA, < nome ... > )

- lo standard **ODMG 2.0** è composto da quattro parti

► **modello ad oggetti:**

- concetti che usiamo per la modellizzazione

► **linguaggio di definizione (ODL):**

- sintassi con cui definiamo i vari costrutti
- definizione dello schema indipendente dal linguaggio di programmazione

► **linguaggio di interrogazione (OQL):**

- specifica come vengono implementate le interrogazioni non procedurali

► **binding a linguaggio di programmazione OO**

- specifica come i costrutti ODL e OQL vengono mappati sui linguaggi di programmazione (C++, java...)

- **modello ad oggetti**

► abbiamo una distinzione tra oggetti e valori letterali

- oggetti

- OID unico nel sistema

- nome [opzionale]: permette di accedere ad uno specifico oggetto

- tempo di vita:

- persistente

- transitorio

- struttura:

- collezione

- atomico: tutto ciò che non è una collezione (anche una struct in questo caso è un valore atomico)

- valori letterali: sono parte dell'oggetto

- valore: conserviamo solo il valore

- non hanno OID

- possono essere

- valori atomici (long, short)

- strutturati struct (tupla???)

- collezioni di:

- oggetti

- valori

► tutto può essere rappresentato come oggetto (anche int, string..) ma questo è molto penalizzante...per questo sono stati introdotti i valori letterali

- concetti di modellizzazione

► interface

► class

► collezioni

- **interface:** definizione di tipo astratta

► definiamo le operazioni che il tipo esporta ma l'interface non è direttamente istanziabile

► **interface primitiva:** definisce i metodi visibili all'esterno che permettono di lavorare sull'oggetto

```
Interface Object {
 boolean same-as (in Object o) //uguaglianza profonda (stesso val e OID)
 Object copy()
 void delete()
}
```

- tutti gli oggetto derivano da questa definizione

- le interface non sono istanziabili ma sono ereditabili (ereditarietà multipla)

- è possibile creare un'altra interfaccia ereditando da più interfacce i metodi disponibili!

- 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 chi disegna l'applicazione vera e propria a definire delle classi

- **class**

- ▶ usata per dichiarare delle classi/tipi definiti da chi disegna il DB
- ▶ rappresenta le definizioni una tupla nel sistema 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 dati all'interno (i dati possono essere collezioni, strutture, valori atomici)
- ▶ la classe ha la struttura: l'indirizzamento dell'oggetto viene implementato in questo modo:
  - dot notation: O1 = O.copy();
  - arrow notation: O1 -> O.copy();

- **collezioni**

- ▶ derivate *interface collection*  
ovviamente non da sapere

```
interface Collection: Object {
 exception ElementNotFound(any element)
 unsigned long cardinality();
 boolean is_empty();
}
```

- ▶ le collezioni che abbiamo a disposizione sono quelle che abbiamo visto in precedenza:
  - 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 una valore data la chiave di ricerca (tavola hash)
- ▶ queste interfacce hanno i metodi di *interface Collection* e una serie di metodi specializzati
- ▶ come si istanziano le collezioni:
  - *set<studente>*
    - questa collezione può contenere solo studenti!
    - una definizione di questo tipo (forte) impedisce la costruzione di categorie

- **extends e extent**

- ▶ se A extends B (dove B è un tipo)
  - A eredita tutti i metodi di B e ne aggiungerà eventualmente altri
- ▶ classe Person extent Persons: avremo una collezione *Persons* (es *set< Persons >*) che contiene tutte le Person --> equivalente di una relazione che contiene tutte le tuple!

- schema OR vs OO:

- **DBMS Object Relational (concetti di base)**

- ▶ proposto come modello a sé stante da alcuni autori, il modello Object-Relational generalizza il modello relazionale incorporando tutta la potenzialità delle basi di dati O\_O
- ▶ Il punto di partenza degli autori è la consolidata tecnologia relazionale di cui si vuole sfruttare non solo il modello di base, ma anche il grande investimento in sistemi oggi disponibili che già mettono a disposizione sofisticate tecniche di distribuzione, gestione della concorrenza, crash recovery, ottimizzazione delle interrogazioni, ecc...
- ▶ uno dei punti di forza della proposta è la compatibilità con le vecchie applicazioni: tutte le applicazioni sviluppate su dbms relazionali devono continuare a funzionare, con modifiche minime, anche sui nuovi sistemi.
- ▶ Il nuovo standard, SQL 1999, è una diretta evoluzione di SQL2; nel 1999 è stata prodotta la prima versione, da cui deriva il nome dello standard
- ▶ Sacco: non abbiamo ancora l'OO completo perché non abbiamo l'ereditarietà

- **note su SQL:1999 (Rispetto all'SQL2 fornisce):**

- ▶ nuovi tipi di dati
  - CLOB (charcter large object)
  - BLOB (binary large object)
- ▶ nuovi predicati di selezione:
  - essenzialmente pattern matching per stringhe (SIMILAR) con l'uso di espressioni regolari.
  - amplia il già noto predicato LIKE
- ▶ ampliamento della semantica (Viste ricorsive);
- ▶ più sofisticati meccanismi di gestione delle transazioni e della sicurezza;
- ▶ adotta la nozione di ruolo, già ampiamente diffusa in vari prodotti;
- ▶ I nuovi tipi di dati si possono categorizzare in:
  - nuovi domini di base e domini utente
  - tavole tipate
    - tuple con OID
    - referenziamenti abbinabili ad attributi
  - tavole in relazione gerarchica
- ▶ funzioni e procedure utente
  - costruzione di funzioni utente che permettono di aumentare le potenzialità offerte dalle funzioni scalari messe a disposizione dal DBMS

- **riassumendo: i tipi delle tavole**

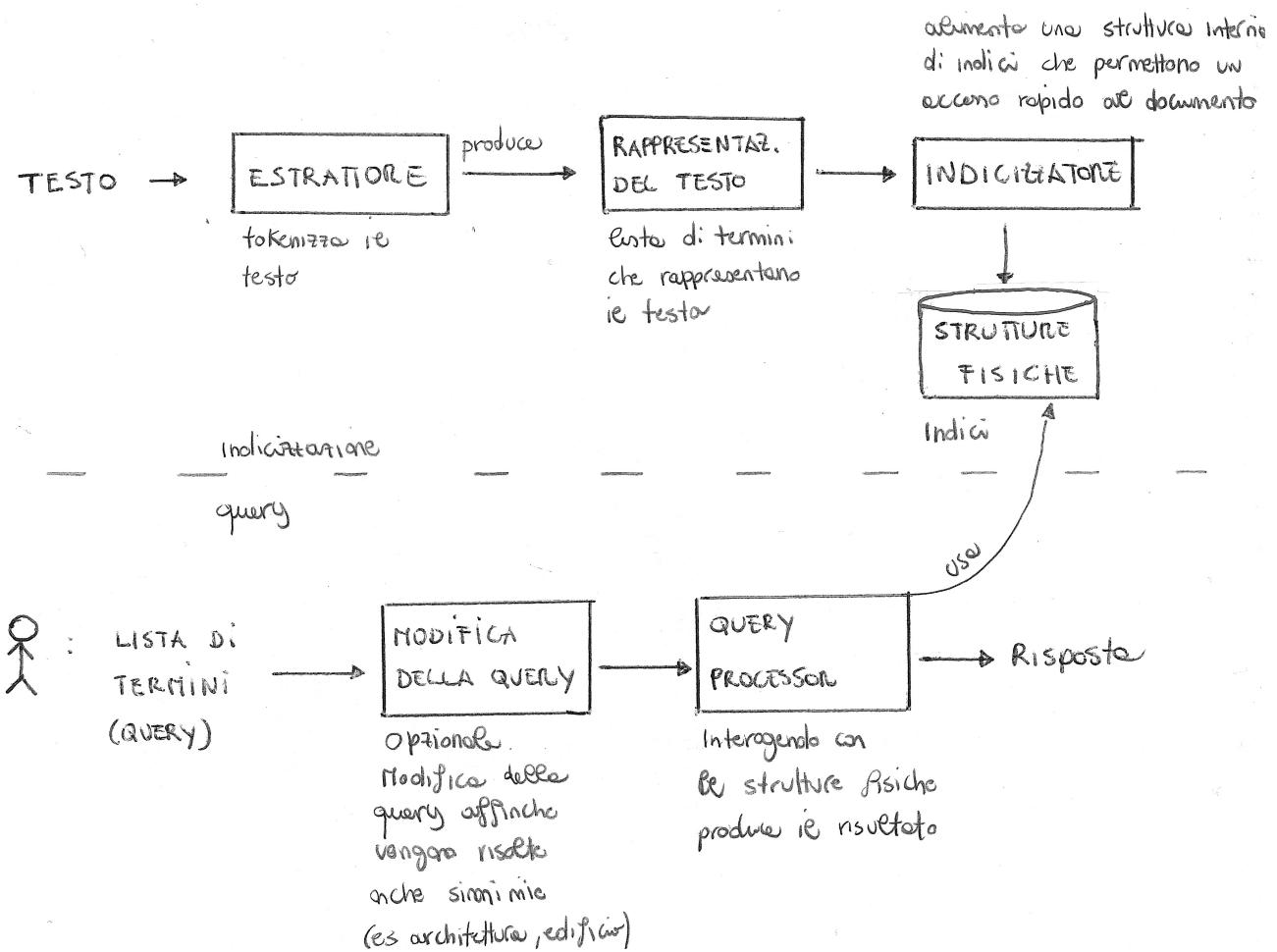
- ▶ possono essere strutturati con uno o più attributi
- ▶ I tipi degli attributi possono essere a loro volta strutturati a qualsiasi livello.
- ▶ l'unica collezione attuale è ARRAY, ma è in studio SETOF.
- ▶ l'utente può definire propri tipi (UDT) che è un potente strumento di estendibilità dei tipi perché un'applicazione può usare indifferentemente un tipo predefinito o un UDT
- ▶ I confronti tra i valori dello stesso tipo utente devono essere definiti da funzioni
- ▶ possono essere strutturati in gerarchie con l'usuale ereditarietà
  - ma un tipo può essere sottotipo di un unico supertipo.
  - segue lo stile Java: non è ammessa l'ereditarietà multipla.

- **Nota sui metodi**

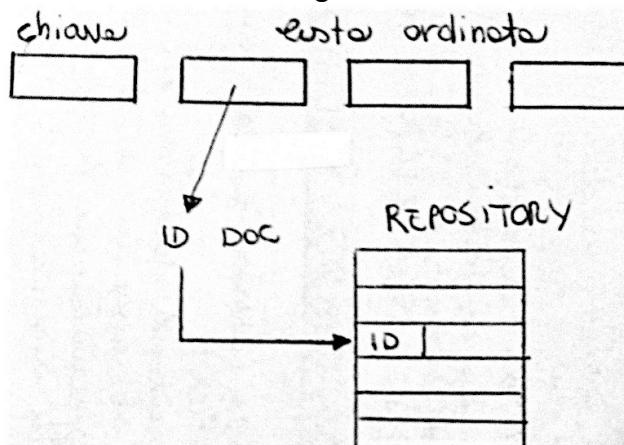
- ▶ i metodi sono strettamente legati ai tipi
- ▶ fanno parte integrante dello schema (interfaccia del tipo utente).
- ▶ per ora la clausola TYPE non prevede sezioni private o protette.
- ▶ non confondere i metodi con le usuali FUNCTION o PROCEDURE dell'SQL introdotte unicamente per gestire valori calcolati o procedure memorizzate. Si ricorda che tali costrutti non hanno parametri impliciti.

## IR - TEXT RETRIVAL

- schema di base di un sistema IR



- tal struttura funziona bene se si cerca una stringa di caratteri (es un nome proprio) ma funziona male se cerco un concetto (es architettura rinascimentale)
- le strutture fisiche sono generalmente gestite tramite **liste invertite**
  - vanno molto bene per risposte binarie
  - in questo caso le liste invertite hanno una **chiave** e una **lista ordinata** che contiene gli identificativi dei documenti che contengono il termine della chiave



- queste liste invertite sono a loro volta gestite da una struttura ad indice: se un utente pone query="cane" riceverà in risposta i titoli dei documenti indicizzati dalla lista invertita che ha come chiave il termine cane

- ▶ dopo che il documento è stato indicizzato, questo non verrà più utilizzato dal query processor per la ricerca (sarà utilizzata la rappresentazione del documento indicizzata)
  - il documento originale può essere restituito come risultato della query
- ▶ la struttura 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 liste invertite siamo in grado di trovare un risultato efficientemente
  - es trovami tutti i documenti che contengono cane??
- problema: enorme **gap semantico** tra il modello del sistema e il modello dell'utente
  - ▶ modello del sistema: si basa sulle stringhe
  - ▶ modello dell'utente: si basa sui concetti
  - ▶ es: supponiamo che l'utente sia interessato all'*architettura del rinascimento*
    - il sistema fornisce tutti i documenti che contengono i termini *architettura and rinascimento*
    - string retrieval! non è esauriente perchè potrei non recuperare tutta una serie di documenti che in realtà potrebbero interessare all'utente
  - ▶ problemi
    - 1. varianti sintattiche inflessionali (sintassi)
      - es: *architetture* e *architettura*
    - 2. sinonimi e concetti semanticamente collegati (semantica)
      - *architettura*
        - ▶ *edificio*
          - *chiesa*
          - *forte*
        - "edificio rinascimentale" è collegato ad "architettura del rinascimento" così come sono collegate allo stesso concetto pure altre cose come "chiesa rinascimentale"
        - 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. la tematica è ignorata nel documento
        - esempi di problema:
          - ▶ i manuali unix non contengono il termine unix!
            - il termine è implicito nel corpus
          - ▶ il termine rinascimento è stato coniato dopo -> tutti i documenti scritti in quel periodo non conterranno tale termine
  - misura usate nell'IR per la valutazione del risultato:
    - ▶ **recall** (esaurienza)
      - capacità del sistema di recuperare tutti i documenti rilevanti rispetto all'interrogazione
      - è la quantità di documenti rilevanti recuperati, rispetto alla totalità dei documenti recuperati
      - *recall = restituiti and rilevanti rilevanti*
        - se recall = 1 ho recuperato tutti i documenti rilevanti
    - ▶ **precision** (assenza di rumore)
      - misura la presenza di documenti non rilevanti nel risultato (rumore)
      - *precision = restituiti and rilevanti restituiti*
        - se precision = 1 ho assenza di rumore (ho recuperato solo documenti rilevanti)
    - ▶ vorremo precision = recall = 1
    - ▶ i sistemi tendono ad avere buona precision ma cattiva recall

► la cattiva recall è dovuta:

- al gap semantico tra modello del sistema e il modello dell'utente
- al fatto che facciamo query in and
  - se facciamo una query e ci vengono restituiti un milione di documenti, l'unica possibilità è quella di mettere in and con qualcosa (in modo da ridurre la cardinalità del risultato)
  - es Q=ADAMO --> 1 milione di documenti (abbiamo molto rumore/ bassa precision)
    - ▶ Q=ADAMO&EVA: perdiamo i documenti rilevanti che non contengono EVA! --> diminuisce la recall!

► considerazioni precision e recall

- sono misure soggettive
- si presume che qualcuno che conosce perfettamente il corpus, a fronte di una query sia in grado di determinare l'insieme di documenti rilevanti e quindi sia in grado di pesare recall e precision in base ai documenti recuperati dal sistema
  - questo è possibile quando si ha a che fare con corpus piccoli
  - risulta invece impossibile calcolare precision e recall con corpus grandi (come il web)
- il problema della recall (esaurienza) 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 corpus molto grandi è utopistico pensare che l'utente vada a vedere tutti i documenti rilevanti
  - dunque la perdita di documenti in DB molto grandi è accettabile, mentre il rumore, soprattutto nei primi documenti è decisamente meno accettabile
- dunque per confrontare due sistemi diversi si utilizza la PRECISION in due punti precisi
  - precision @5: precision complessiva nei primi 5 documenti
  - precision @10
- questo permette di fare valutazioni soggettive ad utenti che non conoscono l'intero corpus
  - è sufficiente che il valutatore (utente) scorra i primi 5 documenti e verifichi quanto sono precisi rispetto all'interrogazione -> in pratica stiamo valutando l'accuratezza del ranking

• termine

- il termine è ciò che decidiamo di indicizzare
- in prima approssimazione usiamo la stringa del documento!
- non tutti i termini devono essere memorizzati (es gli articoli sono irrilevanti dal punto di vista del recupero dei documenti)
- non tutte le parole devono essere indicizzate
- white list
  - c'è una persona che indica quali sono le keyword che sono rappresentative per un documento
  - all'inizio di indicizzavano white list (vocabolario controllato)
  - standardizzazione della rappresentazione del documento
  - nel vocabolario controllato possiamo mettere dei concetti (stringhe corrispondono a concetti)
    - è difficile da mantenere
- stop words / black list:
  - termini non indicizzati (otteniamo un risparmio in spazio senza perdere niente)
  - sono termini troppo generali per avere un'utilità dal punto di vista del reperimento

- legge di Zipf

► è del tipo  $1/x$

►

$$P_i = \frac{c}{i}$$

- $P_i$  = probabilità del termine  $i$ -esimo

- $c$  = costante di normalizzazione

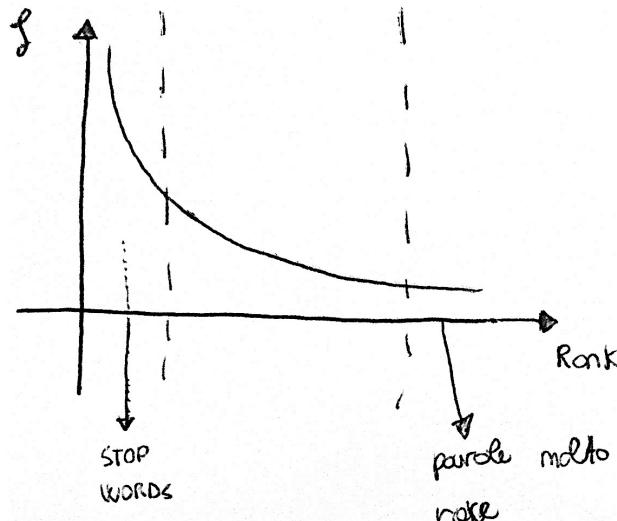
$$\sum_{i=1}^N P_i = 1$$

- essendo probabilità devono essere normalizzate a 1

$$c = 1 + \frac{1}{2} + \dots + \frac{1}{n} = \sum_k \frac{1}{k} = H_n = \ln(N) + \gamma$$

- $H_n$  = numero armonico di ordine  $n$

- $\gamma$  = costante di eulero = 0,577



- problemi

- 1. tokenizzazione
- 2 stop words
- 3 inflessioni
- 4. sinonimi e termini correlati
- 5. termini rilevanti non presenti nel documento

- **1. tokenizzazione**

- cosa usiamo come separatore?
  - spazio?
  - caratteri speciali?
- è possibile che certi termini diventino non ricercabili -> può comportare perdita di recall

- **2 stop words**

- Di Pietro non lo posso cercare! -> perdita di precision

- **3 inflessioni**

- per migliorare la recall dobbiamo considerare tutte le varianti inflessionali di un termine come unico termine! (casa = case = casina ....)
- approcci:
  - riduzione a tema della parola
  - normalizzazione grammaticale

#### ► riduzione a tema della parola

- **stammering**
  - invece di considerare CANE considero CAN
  - 1. il tema non necessariamente rappresenta univocamente una serie di inflessioni che hanno la stesso significato
    - es: tema: NEUTR
    - NEUTRO, NEUTRALE, NEUTRALITA' ... -> ok
    - NEUTRONE -> non va bene
    - il risultato conterà documenti che parlano di neutroni -> riduzione precision!
  - 2. verbi irregolari
    - il verbo ANDARE ha temi diversi: VADO, ANDARE

## ▶ normalizzazione grammaticale

- vogliamo arrivare a una forma grammaticale normale scelta a priori
  - es
    - femminile plurale per aggettivi
    - infinito per i verbi
  - vantaggi:
    - aumento della recall
    - diminuzione delle strutture interne (perchè diminuisce il numero di termini)  
(???)
  - svantaggi:
    - aumento del rumore: es uomo donna (devo sceglierne uno.. ma perdo la differenza) (???)

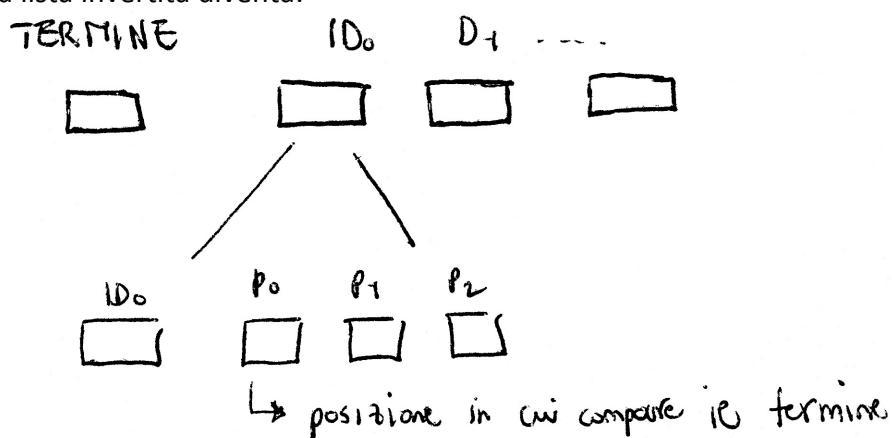
#### • sinonimi e termini correlati

- ▶ la sinonimia e i termini correlati vengono gestiti attraverso thesaurus
  - ▶ il **thesaurus** tradizionalmente identifica il dizionario dei sinonimi
  - ▶ inizialmente i thesauri erano composti solo dai termini con la lista dei sinonimi
    - termine              lista di sinonimi
    - |            | ---> |        | |        | |        | |        | |        | |        |
  - ▶ quello che a noi interessa più che i sinonimi, sono i termini correlati
  - ▶ es di come funzionava una volta:
    - Q = bianco ---modifica della Q---> bianco candido niveo
    - quando l'utente poneva la Q = bianco, il sistema aggiungeva nella query tutti i sinonimi presenti nel thesauro
  - ▶ problema: non necessariamente dei termini sono sempre sinonimi (dipende dal contesto)
    - es il termine CALCIO ha più significati: dal punto di vista della chimica avrà dei sinonimi, dal punto di vista dello sport altri!
  - ▶ recentemente (1998) si utilizzano thesauri organizzati come reti semantiche: WORD NET  
ne è il primo esempio
  - ▶ WORD NET viene utilizzato massivamente per
    - modifica e aumento dell'interrogazione (aggiunta di termini)
    - contestualizzazione dell'interrogazione
    - classificazione automatica dei testi (partiamo da testi grezzi, questi vengono categorizzati dal punto di vista semantico con una classificazione molto precisa; l'utente lavora poi su questa classificazione)

#### • 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

- svantaggi
  - richiede lavoro umano (qualcuno deve leggere il documento e classificarlo)
  - alcuni termini sono ambigui: es STRANIERI può indicare immigrati e giocatori stranieri
  - l'autore del documento può imbrogliare (è lui che sceglie le key word)
- a questo livello consideriamo i termini in qualunque posizione nel documento: stiamo lavorando con il **modello booleano** (SI/NO: ci dice se i termini compaiono o meno nel documento)
  - es: Q = computer and science
    - il modello booleano non tiene conto dell'ordine dei termini né dove sono nel documento
  - in google per fare in modo da considerare la prossimità poniamo Q = "computer science"
  - per considerare la posizione è l'ordine dei termini nella Q gli indici si complicano!
    - nella lista invertita che abbiamo considerato fino adesso non abbiamo nulla che indici la posizione del termine all'interno del documento
    - la lista invertita diventa:



- dobbiamo tenere traccia di tutte le occorrenze di quel termine nel documento con la relativa posizione
- i primi sistemi avevano un overhead delle strutture di indice che era 3 volte superiore al corpus; adesso si arriva ad avere strutture di indici con overhead del 30-50%
- il modello booleano mette allo stesso livello tutti i documenti considerati rilevanti
- noi vogliamo un **ranking del risultato** per rilevanza decrescente

## Ranking

- **relevance ranking principle:** si ordinano i documenti nel risultato per rilevanza decrescente
- problema: come si calcola la rilevanza di un documento rispetto alla query?
  - ▶ ipotesi di bloom: si possono utilizzare i pesi per caratterizzare i termini!
- **schemi di weighting :** servono per stimare la rilevanza dei documenti rispetto alla query
  - ▶ sono basati sulla frequenza del termine nel documento: si assume che i documenti che contengono "X" un numero elevato di volte siano più rilevanti rispetto a quelli che lo contengono poche volte.
- modello **Fuzzy logic**
  - ▶ logica booleana: appartenenza di un elemento ad un insieme: 1 o 0
  - ▶ Fuzzy logic: appartenenza di un elemento ad un insieme: è una probabilità [0-1]
  - ▶ è proprio quello che ci serve: noi vogliamo verificare se un documento appartiene all'insieme dei documenti rilevanti e con che probabilità ci appartiene
  - ▶ maggiore è la probabilità di appartenenza e maggiore è la rilevanza del documento
  - ▶ se usiamo brutalmente la frequenza del termine nel documento ecco come stimiamo la rilevanza:
    - rilevanza  $t = \text{frequenza del termine } t \text{ nel documento } d$

| query         | rilevanza             |
|---------------|-----------------------|
| $t$ (termine) | $ft(d)$               |
| $t$ and $t'$  | $\min(ft(d), ft'(d))$ |
| $t$ or $t'$   | $\max(ft(d), ft'(d))$ |
| $t - t'$      | $ft(d) - ft'(d)$      |

- modello vettoriale

▶ vedi dopo

## • tecniche di weighting

▶ la tecnica di weighting più comune è **TF - IDF** (term frequency - inverse document frequency )

### ▶ TF

- caso più semplice:  $TF = f_{t,d}$  (frequenza del termine  $t$  nel documento  $d$ )
- problema 1:
  - non viene fatta nessuna normalizzazione sulla lunghezza del documento
    - ▶ quindi è un frequenza assoluta?
  - la frequenza dovrebbe essere normalizzata sulla lunghezza del documento
- problema 2 :
  - una stima di questo tipo favorisce i documenti in cui il termine compare più di frequente -> per la Zipf distribution i termini troppo frequenti contano poco
- quello che si usa è (è più in accordo con la nostra percezione):
  - $tf_{i,j} =$ 
    - ▶  $1 + \log_2(f_{i,j})$  se  $f_{i,j} > 0$ 
      - utilizziamo una correzione logaritmica
      - non considera la dimensione del documento
    - ▶ 0 altrimenti
  - ha una crescita meno rapida della  $f_{t,d}$

### - DBM

- $tf = n/K$

▶  $n$  = numero di volta che il termine compare nel documento  
 ▶  $K$  = numero di parole del documento

## ► IDF

- serve per considerare l'importanza del termine all'interno del corpus
- l'idea è di misurare la specificità del termine dal punto di vista statistico (e non semantico)
- con l>IDF si vuole "correggere" la *frequenza* del termine con la *specificità* del termine
- vogliamo tenere conto del fatto che i termini estremamente frequenti comparirebbero in molti documenti e risultano quindi inutili
- vogliamo utilizzare un fattore correttivo perché altrimenti termini di scarso interesse (es articoli) potrebbero comparire con altissima frequenza
- vogliamo quindi favorire i termini più specifici (che compaiono in pochi documenti) rispetto a quelli più generali (che compaiono in molti documenti)
- $\text{idf}_i = \log(N/n_i)$ 
  - N = dimensione corpus = numero totale di documenti
  - $n_i$  = numero di documenti in cui è presente il termine i-esimo
- DBM
  - se un termine è presente in tutti i documenti non è rappresentativo
  - vogliamo che un termine acquisti più valore se contribuisce a differenziare i documenti tra di loro
  - è un fattore correttivo che mi cattura la quantità di informazione che una parola porta rispetto all'intero bacino
  - parola presente in tutti i documenti:  $\text{idf} = \log(1) = 0$

## ► TF-IDF

- la formula di peso diventa quindi:  $\text{tf} * \text{idf}$
- DBM
  - tf-idf: peso che si associa ai singoli termini nella rappresentazione vettoriale del documento considerato
  - la tf viene corretta attraverso idf che ha un andamento opposto
  - è da notare che la *specificità* del documento prescinde dalle considerazioni concettuali
    - modello concettuale:
      - ▶ *bevanda*
        - *birra*
        - ..
      - ▶ ovvero birra è più specifico di bevanda
    - modello statistico:
      - ▶ anche se birra è più specifico dal punto di vista della tassonomia nulla vieta che a livello statistico sia più frequente nel nostro corpus

- **modello vettoriale**

- ▶ prima del modello Fuzzy era stato definito il modello vettoriale (??)
- ▶ 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:  $D(w_0, \dots, w_T)$ 
  - $w_i =$ 
    - 0 se i non appartiene al documento D
    - tf-idf altrimenti
- ▶ query:  $Q(w_0, \dots, w_T)$ 
  - in questo caso la query è solo una lista di termini -> non possiamo utilizzare operatori booleani (?????????)
- ▶ problema: lo spazio vettoriale ha un numero di dimensioni estremamente elevato ( $T =$  numero di termini del dizionario)
- ▶ vantaggio: possiamo definire una misura di somiglianza tra il documento e la query

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

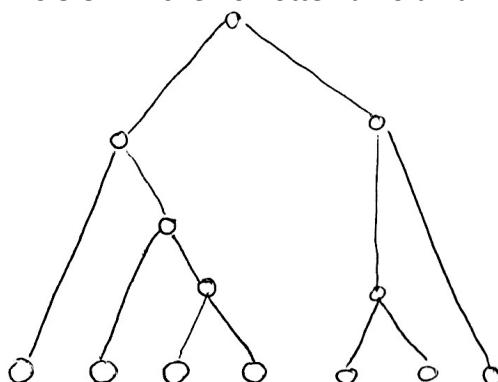
- la somiglianza è data dal coseno dell'angolo tra i due vettori (x e y)
- il coseno dell'angolo è 1 se l'angolo è  $0^\circ$ , mentre è 0 se l'angolo è  $90^\circ$
- ▶ un'altra misura di somiglianza è il **coefficiente di Jaccard**
  - è di tipo insiemistico (non è definito a livello vettoriale)
- $$S(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$
  - numeratore: cardinalità intersezione
  - denominatore: cardinalità unione
  - insiemi uguali:  $S(X, Y) = 1$
  - insiemi disgiunti  $S(X, Y) = 0$
- ▶ nel modello vettoriale abbiamo la stessa rappresentazione per documento e query
  - questo ci permette di misurare la somiglianza non solo tra documento e query ma anche tra documento e documento
- ▶ il problema è che devo calcolare la funzione di somiglianza per ogni documento -> servono enormi risorse di calcolo
  - questo problema può essere risolto con il clustering

## Clustering

- i documenti vengono raggruppati in gruppi omogenei per somiglianza in modo che il confronto venga fatto tra la query e il rappresentante di ogni cluster (e non più tra la query e tutti i documenti del corpus )
- **cluster hypothesis:** documenti simili vengono reperiti insieme
  - ▶ se un documento è rilevante per l'interrogazione, anche i documenti che gli somigliano sono rilevanti
    - ovvero se faccio un interrogazione e recupero un documento D1, recupererò anche tutti i documenti simili a D1
- possiamo raggruppare i vettori dello spazio vettoriale in CLUSTER: vogliamo massimizzare la somiglianza tra documenti all'interno dello stesso cluster e al tempo stesso massimizzare la distanza tra documenti che stanno in cluster diversi
  - ▶  $distanza(x,y) = 1-S(x,y)$
- per ogni cluster scegliamo un **centroide** (un rappresentante) in modo da calcolare la somiglianza tra la query e ogni centroide
  - ▶ se è valida la cluster hypothesis e il centroide di un cluster è interessante per la query -> tutti i documenti del cluster sono interessanti per la query e quindi verranno recuperati anch'essi
  - ▶ la somiglianza è di tipo statistico (non tipo tematico)
    - es:
      - *reato* -> termine prevalente: violenza
        - ▶ *rapina* -> termine prevalente: violenza
        - ▶ *frode* -> non compare violenza
      - i documenti relativi a *rapina* e *frode* possono apparire in cluster diversi, mentre dal punto di vista concettuale l'utente li considera somiglianti
    - per questo il clustering viene principalmente utilizzato per i documenti multimediali
- metodi di clustering:
  - ▶ piatto
    - k-means batch
    - k-means online
  - ▶ gerarchico
    - bisettin k-means
    - dall'alto verso il basso
- **clustering piatto:**
  - ▶ **K-means batch**
    - a priori si sceglie un numero di cluster  $k$  che si vuole ottenere
    - algoritmo
      - 1. selezioniamo  $k$  documenti in maniera casuale -> diventano i centroidi dei  $k$  cluster
      - 2. assegnare ogni documento al cluster con il centroide più vicino
      - 3. aggiornare i centroidi
      - 4. continuare fino a che i centroidi diventano stabili
  - ▶ **K-means online**
    - ha una convergenza più rapida in quanto ricalcola i centroidi ad ogni assegnazione
  - ▶ problemi del clustering piatto
    - è critico scegliere  $k$
    - a causa del punto 1 se ripeto l'algoritmo posso ottenere una clusterizzazione diversa

- **clustering gerarchico**

- ▶ Il clustering gerarchico funziona facendo una gerarchia di dati
- ▶ I dati vengono descritti attraverso un albero tassonomico
- ▶ questo processo si può fare in due modi:
  - per suddivisioni successive: si dividono i dati in due parti, poi ciascuna parte in due parti, poi ciascuna parte in due parti
  - per aggregazioni successive: si uniscono coppie di osservazioni simili, poi si uniscono coppie di coppie, poi coppie di coppie di coppie...
- ▶ Il primo tipo si può giustificare attraverso la teoria
- ▶ non ottengo dei veri gruppi di dati (cluster) come li abbiamo definiti prima, ma solo una tassonomia
- ▶ **bisetting k-means** (dall'alto verso il basso -> spacca)
  - algoritmo
    - 1. assegno tutti i doc ad un unico cluster
    - 2. passo di split
      - ▶ scelgo il cluster più grande
      - ▶ applico k-means con k=2 a quel cluster
    - 3. ripetiamo fino a che il cluster più grande è maggiore di una certa soglia
  - in questo algoritmo fissiamo il numero di documenti massimo che vogliamo avere all'interno di un cluster (e non il numero di cluster come in k-means batch e online)
    - fissiamo la grandezza massima di un cluster, in termini di numero di documenti
- ▶ **dal basso verso l'alto** -> aggreghiamo
  - N = numero di documenti
  - algoritmo
    - 1. ogni documento è un cluster
    - 2. troviamo i due cluster più vicini e li fondiamo
    - 3. ricalcoliamo le distanze tra il nuovo cluster e i vecchi
    - 4. ripetiamo 3 e 4 finché non otteniamo un unico cluster



All'inizio un documento è un 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 due cluster abbiamo un fanout di 2

- **modello probabilistico**

▶ ha portato alla definizione di una serie di stimatori di somiglianza: i BM

▶ BM:

- sta per best match
- sono una variazione sul tema raffinati sperimentalmente
- il più interessante è BM25

▶ BM25

- la similarità tra documento e query è data da:

In cui  $n_i$  è il # di doc selezionati dal termine e in cui  $B_{i,j}$  è :

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

↗  
 Termine  
 ↗  
 doc.

lunghezza  
 del doc*i*  
 ↗  
 avg doc len  
 ↗ lunghezza media  
 dei documenti

↗  
 lunghezza  
 del doc*j*  
 ↗  
 b e K<sub>1</sub> sono parametri  
 empirici definiti di volta  
 in volta. In genere si ha:  
 $K_1 = 1$ ,  $b = 0.75$

- sperimentalmente funziona meglio del modello vettoriale
- è basato su studi empirici
  - la similarità tra documento e query è di tipo empirico

## WEB E MOTORI DI RICERCA

- differenza presente/passato per quanto riguarda l'IR (WEB vs corpus degli anni passati) :
  - ▶ 1. oggi il corpus è molto più grande rispetto al passato
    - oggi il web è più grande di molti ordini di grandezza rispetto a qualsiasi corpus del passato
    - questo comporta un cambiamento dell'architettura
  - ▶ l'incapacità di scalare verso l'alto ha determinato la morte di alcuni motori di ricerca
    - es: altavista è stato scalzato completamente da google nel giro di un anno/due anni
    - google permetteva di gestire meglio l'enorme corpus di documenti rappresentato dal web
  - ▶ 2. il web è un corpus ipertestuale
    - i link possono essere sfruttati per migliorare il risultato
    - questo non succede nel IR classico perchè i documenti sono isolati
  - ▶ 3. il web è ambiente ostile per i motori di ricerca
    - mentre nelle applicazioni precedenti il corpus era omogeneo, e l'unico interesse era quello di permettere di recuperare i documenti, nel caso del web c'è un interesse evidente per i proprietari dei vari siti di comparire prima nel rank
      - le prime posizioni sono quelle che l'utente guarda
    - si deve quindi evitare lo spamming: amministratori che cercano di ingannare i motori di ricerca per far comparire i propri siti nelle posizioni più alte del rank
    - si è passati da una situazione in cui era l'amministratore del sito che specificava le key word di ricerca del sito (era molto facile ingannare i motori di ricerca) a una situazione in cui c'è una guerra tra motori di ricerca (che cercano di evitare spam) e proprietari di siti che cercano di ingannare i motori di ricerca per aumentare la visibilità
    - il nuovo page rank non è molto pubblico perchè il vecchio page rank stava per essere sovrapposto a causa dello spam
  - ▶ 4. i motori di ricerca costituiscono un immensa fonte di guadagno
    - tutto ciò che riguarda l'interazione con l'utente sta assumendo un ruolo molto importante
    - mentre nei sistemi di ricerca precedenti, quello che si dava era un box di tipo testo basilare, adesso invece abbiamo una interfaccia utente molto più completa
    - google addirittura ci suggerisce delle query
      - questo permette di risolvere uno dei problemi dell'utente nell'IR: cosa chiedo? che parole devo usare?
      - abbiamo quindi un assistenza alla ricerca!

### Ranking

- dare un rank (una classifica) ai risultati (documenti) trovati: in modo da avere in cima alla lista dei risultati i documenti più rilevanti
- è la funzione più importante di un motore di ricerca
- il modello booleano non permette nessun tipo di rank
- problemi
  - ▶ evitare lo spam: ovvero che amministratori cerchino di aumentare con escamotage il rank delle proprie pagine
    - effetti negativi dello spam:
      - è un problema per l'utente
      - motori di ricerca fanno soldi con le pubblicità! -> pagine spam con alto rank limitano i guadagni
  - ▶ una volta che abbiamo definito una funzione di ranking, questa deve anche essere calcolata... trade off tra

- semplicità di calcolo
- accuratezza
- contenuti informativi da tenere in considerazione per il rank (oltre alla somiglianza con la query):
  - ▶ nomi di domini in cui si trova una pagina: es la voce di wikipedia compare per prima
  - ▶ link: page rank permette di pesare i link
  - ▶ altri dati:
    - titolo
    - metada: key word, che però solitamente vengono ignorate perchè possono essere usate impropriamente
    - font size più grande indica che quella parte è più importante
    - titoli schemi di peso
  - ▶ BM25
  - ▶ segnali strutturali
    - come la pagina che stiamo esaminando si pone nei confronti delle altre pagine (link in entrata, in uscita...)
    - page rank!
  - ▶ monitorare i click through:
    - il fatto che l'utente clicchi un link e poi torna subito indietro vorrà dire che la pagina a cui ha acceduto era di scarso interesse
    - questo ci permette di migliorare il rank di specifiche pagine
  - ▶ contesto geografico:
    - se sono in Italia sono più interessato a pagine in italiano
    - se sono a Torino e cerco un bar ...
  - ▶ contesto tecnologico e temporale:
    - il fatto che usi un mac piuttosto che windows può dire qualcosa..
    - è complicato da usare
  - ▶ contesto temporale:
    - monitorare la storia interrogazioni
    - è complicato da usare
  - ▶ considerare l'architettura di link che abbiamo a disposizione (segnali strutturali)
    - è un aspetto del tutto nuovo rispetto all'IR classico
    - nel web i documenti sono collegati tra di loro tramite link
      - possiamo sfruttare questi link per determinare il rank
    - quello che ci interessa calcolare è il livello di autorevolezza della pagina:
      - a fronte di una query vogliamo restituire per prime le pagine più rilevanti rispetto alla richiesta ma anche le più autorevoli
        - ▶ es di wikipedia
      - lo scopo è quello di dare all'utente l'informazioni più utile sia dal punto di vista della rilevanza (con la query) sia dal punto di vista di contenuto autorevole
      - misurare l'autorevolezza di una pagina P semplicemente contando il numero di link in ingresso a quella pagina è un meccanismo che può essere facilmente ingannato
        - ▶ creo tante pagine che puntano a P
      - per correggere questo meccanismo è stato proposto page rank

## Page rank

- la versione che vedremo è comunque ormai superata, in quanto è diventata vulnerabile tramite meccanismi di link incrociati detti link farm
- situazione: c'è un nemico, lo spammer, che cerca di ingannare il motore di ricerca attraverso meccanismi sempre più sofisticati
- il page rank viene calcolato nel momento in cui i documenti sono indicizzati
  - ▶ ovvero nel momento in cui il web viene attraversato da programmi che si chiamano crawler
  - ▶ questi robot vanno in giro per il web, prendono le pagine che incontrano e le indicizzano
- lo scopo è trovare la probabilità che un utente (il quale si muove casualmente nel web) si trovi in una determinata pagina
  - ▶ questa probabilità è il page rank!
  - ▶ page rank di una pagina  $a$ :

$$PR(a) = \frac{q}{T} + (1 - q) \sum_{i=1}^n \frac{PR(p_i)}{L(p_i)}$$

- $T$  è il numero totale delle pagine del web
- $q$  è un parametro settato dal sistema (tipicamente 0,15)

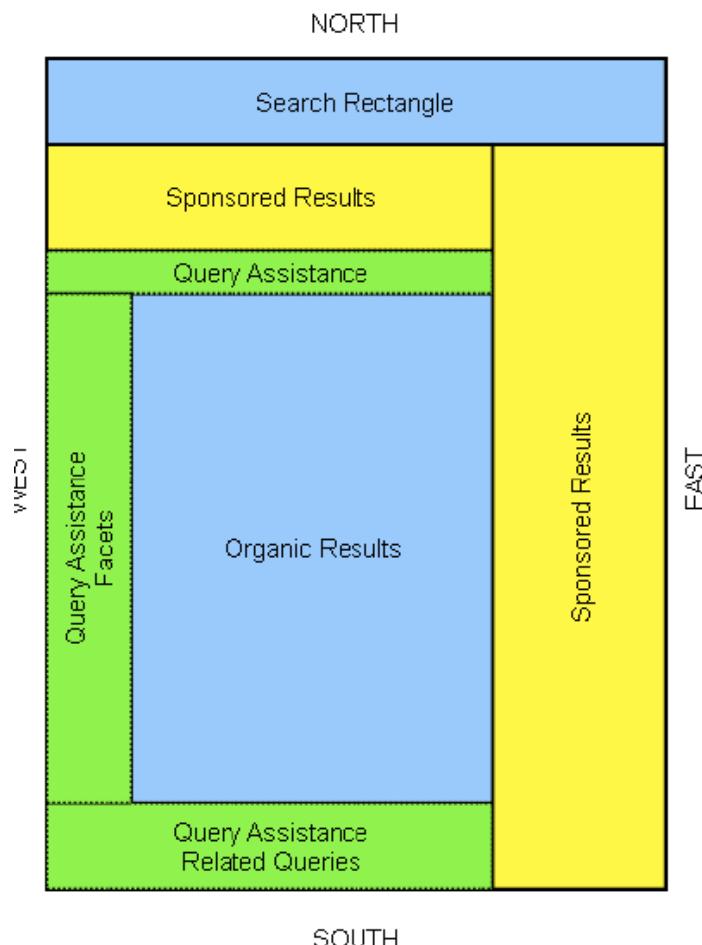
- il page rank è totalmente indipendente dalla query
  - ▶ per questo solitamente il page rank viene combinato con altri meccanismi di valutazione (quelli di pertinenza con la query)
    - es combiniamo page rank con BM25!
    - possiamo avere meccanismi più complessi, come già visto, ad esempio considerando font-size.... ecc ecc
- Page Rank DBM
  - ▶ l'idea alla base è che il meccanismo di navigazione all'interno del web possa essere visto come un meccanismo di surfing random, ossia data una pagina  $P$  la probabilità di seguire uno degli  $n$  link presenti in  $P$  è la stessa
    - se una pagina ha 4 link, la probabilità di seguirne uno è proporzionale a  $1/4$
    - diciamo "è proporzionale" e non "è" perché occasionalmente l'utente digita direttamente un URL (lo fa con probabilità bassa:  $1-\beta$ )
    - se  $1-\beta = 0.05$  allora la probabilità di seguire uno di questi 4 link è  $(1/4) \cdot (0,95)$
  - ▶ page rank di  $P$  = probabilità che il surfer ad un certo punto si trovi in una pagina  $j$ :

$$P(j) = \frac{1-\beta}{N} + \beta \sum_{i \in in(j)} \frac{P(i)}{out(i)}$$

- $P(j) = A + B$ 
  - ▶ probabilità che l'utente sia nella pagina  $j$  = page rank della pagina  $j$
  - ▶  $A$  probabilità che l'utente sia arrivato alla pagina  $j$  in modo casuale
  - ▶  $B$  probabilità che l'utente sia arrivato alla pagina  $j$  seguendo un link
- $P(i)$ 
  - ▶ page rank della pagina  $i$  = probabilità di seguire un link all'interno della pagina stessa
- $out(i)$ 
  - ▶ è il numero di archi in uscita a  $i$
- $in(j)$ 
  - ▶ insieme degli archi entranti in  $j$
- ▶ abbiamo un modo per definire l'importanza delle pagine tenendo conto solo della struttura delle pagine

----- end page rank -----

- gli ultimi algoritmi usati da Google seguono una strada differente dal page rank: si utilizzano degli algoritmi di **machine learning** per valutare quali pagine sono rilevanti e quali sono spam
  - ▶ la tecnica è abbastanza interessante
    - si parte dal fatto che i siti spam sono visibilmente e strutturalmente diversi da quelli reali: es vogliono vendere A ma parlano di B
    - vengono selezionati una serie di siti in maniera casuale e vengono valutati in maniera manuale
      - problema di scalabilità:
        - ▶ bisogna avere migliaia di persone
        - ▶ turco meccanico di Amazon: migliaia di persone che svolgono questo lavoro per poco tempo e a basso costo
- la **valutazione degli algoritmi di ranking** viene fatta solitamente in maniera semiautomatica:
  - ▶ misuriamo la precisione a diversi livelli di risultato: @5, @10, @20 attraverso:
    - lavoro manuale (es turco meccanico)
    - oppure approssimiamo la valutazione sulla base dei click degli utenti:
      - ci si basa sul click trough: quanto l'utente rimane su una pagina del risultato?
      - se l'utente nei primi 5 link entra e subito torna indietro la precision @5 sarà bassa!
- componenti recenti:
  - ▶ snippets
    - sono le 2/3 righe di testo che compiono sotto i titoli dei risultati Google
    - solitamente nello snippets vengono evidenziate le parole che compaiono nella query utente
    - è importante la qualità dello snippets: convincerà l'utente a cliccare o meno sul link!
  - ▶ meccanismi di suggerimento della query per aiutare l'utente
- layout base di un motore di ricerca



# Tassonomie dinamiche

La ricerca tradizionale è basata sul **recupero delle informazioni**.

La maggior parte dei compiti comuni però prevede un'**esplorazione**

- trovare delle relazioni
- diminuire le alternative

Si pensi ad esempio ad un utente che vuole comprare una macchina fotografica ma non sa esattamente quale; comincerà innanzitutto a valutare il parametro prezzo, fatta la scelta (es. prezzo basso) il numero di macchine risultanti sarà diminuito, a questo punto può continuare in questo modo scegliendo altri parametri di interesse (sensibilità iso, pixel ecc.) e continuando così a sfoltire le possibili scelte finché non arriverà ad un insieme di fotocamere che soddisfino la sua scelta.

All'utente è richiesto di:

- trovare tutte le possibili caratteristiche
- pesare queste caratteristiche e focalizzarsi sulla più rilevante
- esplorare e trovare tutte le caratteristiche correlate
- ripetere il processo finché il numero di elementi selezionati è sufficientemente piccolo per un'ispezione manuale

Rappresentazione

- parte intensionale: il database è rappresentato da una tassonomia disegnata da un esperto
- parte estensionale: i documenti possono essere classificati ad ogni livello di astrazione ed ogni documento è **classificato sotto due o più concetti**. (Per documenti intendiamo qualunque oggetto, non solo testuali)

Un concetto è un'etichetta che identifica un insieme di documenti classificati sotto tale concetto.

Nessuna relazione a parte la sussunzione (IS-A, PART OF) deve essere mantenuta nello schema. La sussunzione richiede di mantenere un **vincolo di inclusione** per cui: se  $D(C)$  denota un insieme di documenti classificati sotto  $C$ , e  $C'$  è un discendente di  $C$  nella gerarchia allora →  
$$D(C') \subseteq D(C)$$

Come sono correlati tra loro i concetti?

- tramite **sussunzione** (is-a, part-of)
- tramite regola di **inferenza estensionale**: due concetti  $C$  e  $C'$  sono correlati se c'è almeno un documento classificato sia sotto  $C$  che sotto  $C'$  o uno dei loro discendenti. Risultato importante → una relazione tra due elementi non rappresentata esplicitamente nello schema viene inferita sulla base di un'evidenza empirica

Idea della tassonomia dinamica:

1. settare il focus di interesse
2. riassumere i concetti correlati ed eliminare quelli non correlati

Conseguenza importante: Le relazioni tra concetti non devono essere anticipate ma possono essere inferite dalla classificazione attuale.

Vantaggi:

- schema più semplice
- si adatta a nuove relazioni (dinamico)
- si scoprono delle relazioni inaspettate

Il termine **“tassonomie dinamiche”** è usato per indicare che la tassonomia può adattarsi a, e riassumere ogni sottoinsieme dell'universo, mentre le tradizionali tassonomie statiche possono riassumere solo l'intero universo.

**Benefici (relativi anche all'esempio di ricerca con zoom delle slide):**

- interfaccia più semplice e famigliare
- l'utente è effettivamente guidato alla raggiunta del suo obiettivo. Ad ogni stadio ha una lista completa dei concetti correlati
- trasparenza: l'utente sa esattamente cosa sta succedendo
- interazione completamente simmetrica: se A e B sono correlati, l'utente troverà B facendo lo zoom su A e troverà A facendo lo zoom su B
- scoperta di relazioni inaspettate
- nessun risultato vuoto
- ogni combinazione di concetti (AND, OR, NOT) è supportata dalle corrispondenti operazioni insiemistiche sulle loro estensioni profonde
- supporto multilingua semplice: (basta tradurre le etichette)
- semplice raccogliere informazioni sull'interesse dell'utente
- schema semplice e minimale: non sono necessari argomenti composti. Es. quadri in Lettonia non deve essere un concetto perchè si può derivare dall'intersezione tra Latvia e quadri (ottenuti a runtime).
- Integrazione con altre tecniche di recupero (information retrieval, database):
  1. è possibile dare un contesto concettuale grazie alla tassonomia dinamica e poi sugli elementi rimasti applicare una normale tecnica di ricerca
  2. è possibile eseguire una normale ricerca e grazie alla tassonomia dinamica riassumere concettualmente i risultati ottenuti da tale ricerca.  
(possono essere combinati)

## Linee guida per il disegno di tassonomie

Organizzare la tassonomia come un insieme di sottotassonomie **ortogonalmente indipendenti**. Queste sottotassonomie indipendenti sono generalmente chiamate “facets”.

Avvertimento: stare attenti alle **false correlazioni**, che possono nascere da documenti composti (documenti che non sono atomici, ma che contengono altri documenti)

In ogni caso l'eliminazione di concetti correlati è una linea guida, non un obbligo. Quando dei concetti correlati sono conosciuti e largamente usati, si possono tranquillamente inserire (es. usare internet e non sintetizzare con: computer e network)

**fanout:** il numero di figli per ogni concetto non dovrebbe andare oltre 10-20

**profondità:** la tassonomia non dovrebbe essere più profonda di 3-4 livelli

Riassumendo quanto detto finora, i punti fondamentali della tassonomia dinamica sono:

- un modello minimale ancora effettivo. Gli utenti lo capiscono
- la regola di inferenza estensionale definisce un meccanismo per adattare la tassonomia al variare del focus dell'utente
- il modello è basato sui metadati. I documenti che devono essere gestiti non sono necessariamente testuali
- chiara separazione tra classificazione ed uso (esplorazione)

## Analisi della convergenza

Tassonomie **monodimensionali**: massimo raffinamento a livello terminale. Numero di terminali un ordine di grandezza minore del corpus

Tassonomie **multidimensionali**:

1. **senza composizioni** di concetti: massimo raffinamento a livello terminale. Il numero di terminali richiesti può essere più grande del corpus (peggiore delle tassonomie monodimensionali)
2. **con composizione AND** di concetti
  - **con classificazione a facet:** decisamente migliore delle altre. Es. una tassonomia con 1.000 terminali e 10 facets è in grado, con 3 operazioni di zoom di passare da 10.000.000 di documenti a 10. (non è una sensazione psicologica. È realmente più veloce!!)

Le tassonomie dinamiche hanno un range di applicazioni molto ampio, es: siti di e-commerce, cataloghi online, brokeraggio di lavoro, sistemi di diagnostica ecc.