

# MODELLI AVANZATI

## Introduzione al corso

### Modelli avanzati

- Strutturato
  - Object-oriented (Rivoluzionario): oggetti persistenti e possono essere recuperati e condivisi da altri programmi
  - Object-relational (Evolutivo): aggiunge al modello relazionale costrutti OO
- Non strutturato (Information retrieval per search engine) indirizzano grossi moli di dati non strutturate
  - Smart-Salton
    - Vettoriale: Doc e query sono rappresentati da uno spazio t dimensionale
    - Booleano: Doc e query sono rappresentati da un insieme di termini di indicizzazione
    - Deviazione della casualità
  - Valutazione di un search engine
    - Recall, tutto ciò che è rilevante viene recuperato, esaurienza del risultato.
    - Precision, solo ciò che è rilevante viene recuperato, assenza di rumore nel risultato
  - Web
    - Crawlers / Robot, che visitano tutte le pagine dei siti, trovano le modifiche e aggiornano la banca dati
    - DB Multilingua,
    - Page ranking, favorisce le pagine che hanno il numero maggiore di riferimenti
    - Autocomplete
- Semi-strutturato (Tassonomie dinamiche: Il prof ha scritto un articolo nel 2000 (leggere)), permettono un accesso guidato alle informazioni. Una tassonomia è una struttura concettuale che va da un concetto più generale ad uno più specifico. La tassonomia viene utilizzata per supportare una classificazione multidimensionale dei dati che possono essere quindi classificati sotto più padri.
  - Faceted search, elimina dalla tassonomia tutti i rami che non sono utili per la ricerca perchè porterebbero ad un risultato vuoto, in modo da semplificare notevolmente la ricerca.
  - Modalità INTENSION: il sistema dinamico torna indietro nei documenti per recuperare concetti usati per classificare ciò che ho recuperato
  - Modalità EXTENSION: Il sistema adatta alla tassonomia il focus di interesse eliminando dei rami inutili dalla ricercaCon questi metodi è possibile raffinare ulteriormente iterando.

### Architetture

Tratta di come un database funziona al suo interno.

Dispositivi di memorizzazione, come sono fatti i dischi, i raid e i dispositivi a stato solido.

- Query, (selezione, proiezione e join)

- External sort
- Piani di esecuzione
  - Nested loop, basati sul prodotto cartesiano
  - Merging scans, basati sull'ordinamento (richiedono che le due tabelle siano ordinate in base all'attributo di join)
  - Frammentazione (Recursive hash partitioning), basato su una tecnica divide and conquer.
- Tecniche di ottimizzazione, utili all'ottimizzatore per trovare il migliore piano di esecuzione. Ci sono tecniche di ottimizzazione esaustive e euristiche.
  - Tecniche banali, anticipazione delle proiezioni, etc...
  - Semijoin, serve solo per ridurre l'insieme di join. Può essere realizzato usando tecniche probabilistiche (ad esempio il filtro di bloom). Il filtro di bloom non elimina però tutte le tuple da eliminare.
- Gestione dei buffer, è simile alla paginazione in sistemi operativi con sistemi a memoria virtuale. Vedremo un modello primivo, chiamato hot set model, che prevede la dimensione di una query in base alla dimensione del buffer. Possiamo farlo perchè vengono utilizzati linguaggi non procedurali (non è il programmatore a decidere come eseguire le operazioni, ma è il sistema).
- Indici, vedremo sia i B-tree (costo logaritmico) che l'hashing estendibile (costo costante). Una prima divisione è fatta in base alla tipologia di indice (primario o secondario), mentre una seconda classificazione è fatta in base a clusterizzazione e molteplicità dei valori.
- Disegno fisico di database, unisce tutto quanto visto prima per realizzare db complessi
- Transazioni
  - Gestione dei malfunzionamenti
  - Controllo della concorrenza

## Introduzione ai modelli classici

### Introduzione

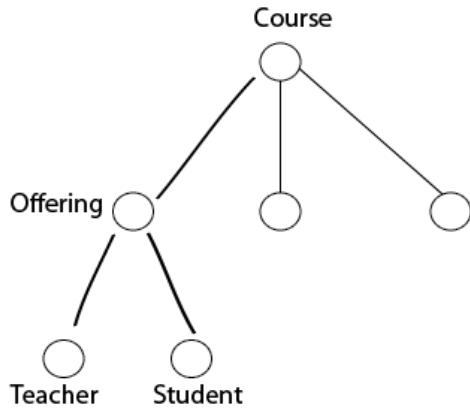
'60 – Modello gerarchico, si poteva rappresentare solo associazioni 1:n

'70 – Modello reticolare (network), basato su puntatori. I problemi principali sono la complessità, la non indipendenza fisica (il programmatore è consapevole) e la proceduralità.

'80 – Modello relazionale. E' semplice dal punto di vista concettuale e dal punto di vista della manipolazione. La critica più importante a questo modello è quella fatta da Kent che individua una disomogeneità orizzontale e verticale.

### Modello gerarchico

I dischi erano poco frequenti e venivano usati i nastri magnetici, che avevano quindi un accesso sequenziale. Il modello gerarchico, a partire da una radice posso avere molti figli (modello parent-child). Questa gerarchia presenta molti vantaggi dal punto di vista fisico (per memorizzarlo) ma presenta molti svantaggi dal punto concettuale e logico.



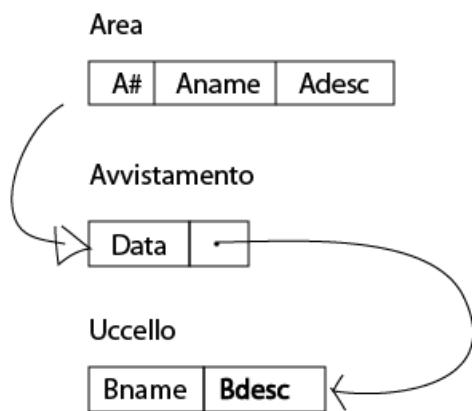
## Svantaggi

- Le relazioni rappresentabili sono soltanto 1:n.
- Asimmetria: tutto va bene se parto da course, mentre se sono interessato agli insegnanti devo scorrere tutto l'albero. L'asimmetria comporta che ho un ordine di accesso predeterminato. Alcune query sono risolte bene, altre molto male.
- Anomalie: è impossibile avere uno studente se non sono iscritto a qualche corso.

Il **primo problema** è quello più delicato; per risolverlo il modello gerarchico si è evoluto (in modo poco elegante). La prima soluzione trovata è di duplicare i dati, mettendo teacher come radice del nuovo albero. Questo non va bene perché lo storage era molto costoso. Successivamente sono stati utilizzati sistemi di puntamento tramite pointers; invece di duplicare i dati, vengono create più gerarchie collegate tra di loro tramite i pointers. Per la **risoluzione della anomalia** la modalità è quella della duplicazione dei dati: si costruisce una nuova gerarchia con una diversa radice. Dunque si hanno più gerarchie con radici diverse.

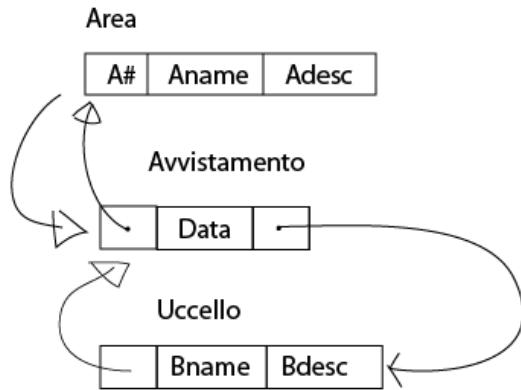
## Esempio

Abbiamo un database per memorizzare le osservazioni di uccelli in determinate aree. Lo schema di navigazione è il seguente:



La navigazione parte da un'area e arriva agli uccelli. Il cammino d'accesso è unidirezionale, non si può partire dagli uccelli per arrivare all'area.

Si usa la tecnica del **virtual pairing** che si basa sull'uso di associazioni logiche bidirezionali. Questa tecnica rende tutto navigabile in maniera arbitraria, complicando però la struttura.



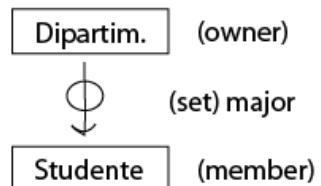
E' come se avessi due gerarchie senza però dover duplicare i dati.

Cambiando i supporti di memorizzazione, da accesso sequenziale ad accesso diretto, è possibile utilizzare i puntatori. I puntatori sono una "pezza" e complicano notevolmente la struttura.

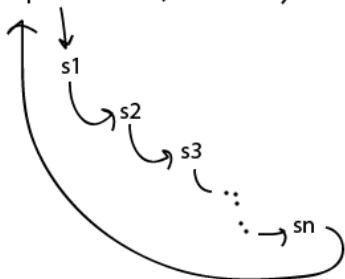
### Modello reticolare

Negli anni 70 viene proposto un nuovo modello, il modello reticolare (network data model) dal task group DBTG e propone uno standard CODASYL. Non è mai stato implementato nella sua interezza da nessun sistema commerciale. È basato pesantemente sui puntatori. L'unità, come nel caso precedente, è il **record**. Abbiamo dei **tipi di record** e possiamo rappresentare delle relazioni n:n. Le relazioni partono comunque dall'1:n, definendo una struttura SET.

L'immagine è un esempio di relazione 1:n.



comp. science → human → biology



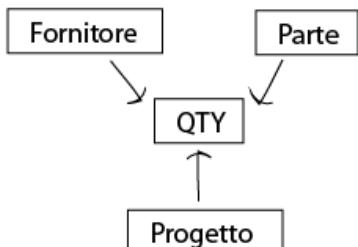
Questa struttura è visibile a livello di transazioni (e quindi al programmatore) che naviga in questa struttura a puntatori.

Possiamo avere dei **multimember sets**: oggetti diversi possono partecipare alla stessa associazione. Ossia i membri di un associazione possono essere di tipi diversi. E possiamo avere **set ricorsivi**. Abbiamo inoltre la possibilità di definire a livello di sistema, dei **set speciali** che non hanno owner e il cui owner è il sistema vero e proprio.

Lo special set serve solitamente per numerare tutti gli elementi di un gruppo, ad esempio tutti i dipartimenti. Scelto poi un singolo dipartimento si scende per vedere i suoi studenti.

Come possiamo realizzare relazioni n:n?

Non siamo in grado di rappresentarlo con un set. Il meccanismo utilizzato è quello del linking "dummy record type", che funziona come member di più owner.



E come fai con le 1:1? Si implementa con un vincolo.

Il problema, oltre a essere molto macchinoso, è che è completamente procedurale.

L'interazione del programmatore segue i puntatori e quindi naviga completamente il database.

### Modello relazionale (1970)

Quando viene proposto il modello relazionale, otteniamo un modello di rappresentazione estremamente semplice basato su tabelle che possono rappresentare entità oppure associazioni. Abbiamo soltanto un meccanismo di puntatori logici (chiave primaria e foreign key) invece che avere puntatori fisici. Il modello di Codd semplifica molto anche la realizzazione delle applicazioni perché definisce un metodo matematico non procedurale per definire il risultato di un'operazione demandando al sistema il piano di esecuzione. Passiamo ad un'interfaccia basata su insiemi, invece che su record.

#### Vantaggi:

- Semplice ma completo
- Non procedurale
- Separazione logico-fisica

Le difficoltà incontrate dal modello relazionale sono principalmente due: molte persone avevano adottato il modello reticolare e la transizione non è banale, l'altro motivo è che i primi sistemi relazionali erano lenti.

#### Svantaggi:

I problemi del modello relazionale vengono evidenziati da W. Kent nell'articolo "Limitations of record-based information models" del Marzo 1979 (da leggere). La critica di Kent si concentra sui modelli basati su record.

### Osservazioni di Kent sul modello relazionale

E' la critica più importante al modello relazionale. Un record è una sequenza fissata di campi che sono conformi a una descrizione statica, quindi fatta a priori. Le obiezioni di Kent ad una struttura di questo tipo sono due (chiamate **disomogeneità**):

#### Disomogeneità orizzontale

Se noi organizziamo una sequenza fissa di campi, o mettiamo tutti i campi possibili, ma molti campi rimangono null o non riusciamo a descrivere completamente l'elemento (ad esempio per un capo di vestiario ha diversi campi a seconda se è un pantalone o una maglia). La soluzione è avere una gerarchia di tipi con generalizzazioni e specializzazioni.

## Soluzione

Risolviamo la disomogeneità orizzontale introducendo dei **reticoli semantici**:

??img9

In questo caso abbiamo due caratteristiche:

1. Ereditarietà dei dati: Studente-Lavoratore eredita le informazioni sia da Studente che da Lavoratore.
2. Proprietà di inclusione sulle istanze (subsumption):  $A < B$  ( $B$  sussume  $A$ ) implica che le insieme delle istanze di  $A$  sia un sottoinsieme, probabilmente improprio, dell'insieme delle istanze di  $B$ . Le istanze di Studente sono un sottoinsieme delle istanze di Persona.

Se appiattiamo la struttura, otteniamo l'equivalente di trattare tutti come Studenti-Lavoratori. Avremo la struttura Persona(CF, nome, università, matricola, società, stipendio). Se abbiamo uno Studente-Lavoratore utilizziamo tutti i campi, ma negli altri casi avremo soltanto alcuni campi utilizzati mentre gli altri vengono lasciati a null.

Un modello in grado di rispondere alle critiche di Kent deve supportare dei reticoli semantici. Quello che si deve supportare è il concetto di ereditarietà multipla che serve a modellare di casi che esistono in realtà in maniera semplice. Invece di avere una realtà ben definita possiamo ereditare in maniera multipla da due oggetti della gerarchia e definire un nuovo oggetto. Oltre all'ereditarietà degli attributi una struttura di questo tipo impone dei vincoli (SUBSUMPTION). A tali vincoli è legato il concetto di estensione superficiale e estensione profonda.

## Estensione superficiale e profonda

Estensione superficiale lo studente dal punto di vista superficiale sta nell'entità studente.

Estensione superficiale e profonda sono strettamente collegate al concetto di sussunzione.

## Disomogeneità verticale

In un tipo di record un campo contiene sempre lo stesso tipo di valore. Ovvero abbiamo tipi diversi come valori di uno stesso campo. Questo non va sempre bene, perché ci sono situazioni in cui abbiamo oggetti diversi che partecipano ad una qualche associazione.

Pensiamo ad esempio ad un'azionista di una società: può essere una persona fisica oppure una persona giuridica. L'associazione è la stessa ma dovrebbe essere rappresentata in modo diverso perché l'oggetto coinvolto è di tipo diverso.

## Soluzione

Risolviamo il problema della disomogeneità verticale tramite l'**identificazione delle istanze**.

Il modo in cui identifichiamo un record nel modello relazionale è tramite la chiave primaria che deve soddisfare il *vincolo di unicità*.

Ottieniamo diversi problemi:

1. Non uniformità: le chiavi primarie sono diverse da relazione a relazione. Ad esempio la chiave primaria delle persone fisiche è diversa dalla chiave primaria delle persone giuridiche.
2. Non persistenza: se il mio codice fiscale cambia, tutte le relazioni devono essere aggiornate.

Per risolvere il problema viene utilizzato un **surrogato** il cui unico scopo è di identificare un'istanza senza fare riferimento ai dati interni dell'istanza. Tale surrogato è un **identificatore opaco** definito dal sistema che è persistente, ovvero è invariabile per tutta la vita dell'istanza. Tipicamente è un numero di 64 bit.

### **Architettura record-based**

Due punti fondamentali dell'approccio per risolvere i problemi trovati da Kent sono:

1. Gerarchie di tipi
2. Identificazione tramite surrogati

Quando parliamo di **record** abbiamo in mente una memorizzazione contigua delle informazioni. Vediamo il record come aggregazione fisica, ma questo è un problema.

- Una caratteristica di tipo fisico viene usata a livello logico: si ha un impatto negativo nelle prestazioni del sistema. In una relazione con 100 attributi accedo sempre a un numero limitato di essi, una trazione dal punto di vista fisico accede sempre all'intero record, quindi a tutti gli attributi.
- Un cambiamento della descrizione nei dati comporta una ristrutturazione della relazione
- Problemi semantici: Part of. L'aggregazione modella una relazione che è fatta di parti ma spesso gli attributi non sono una parte dell'entità

Spesso però alcuni campi non sono una parte di un'entità ma spesso sono un'associazione ad un'entità (ad esempio l'età non è una parte di una persona, ma è associata ad una persona). Il concetto di record come aggregazione fisica di campi non è consigliato a livello logico, ma può anche essere dannoso a livello fisico perché comporta un numero troppo elevato di "dati" memorizzati in una stessa "tabella".

Il concetto di record (che comunque persiste ad oggi) è fondamentalmente un concetto che deriva dall'attività di programmazione object-oriented. Quando lavoriamo in memoria secondaria però questo non è consigliabile.

### **Problemi dei record**

Gli attributi di un record si dicono *caldi* se vengono utilizzati spesso e *freddi* se vengono utilizzati raramente. Ad ogni accesso recuperiamo tutto il record anche se non abbiamo bisogno di tutti gli attributi.

Un altro problema è che abbiamo delle strutture instabili. Se modifichiamo la struttura della relazione dobbiamo ristrutturarla interamente.

Anche se vedremo strutture basate su record, questa non è un'ipotesi fondamentale.

?? mettere OQL

Possiamo cambiare la rappresentazione interna senza cambiare la struttura del programma in quanto le variabili interne sono accessibili solo attraverso metodi. Prototipo azione più semplice scelgo delle implementazioni banali per far vedere che il sistema funziona e successivamente raffino tali implementazioni.

Intorno agli anni 70 smalltalk introduce l'idea di gerarchia dei tipi e quindi l'ereditarietà dei dati e delle operazioni. La gerarchia dei tipi semplifica sì arrivo uso il disegno incrementale in

quanto permettono disegno per affinamenti successivi. Motivazione di Ing. Del software la separazione tradizionale tra linguaggi di programmazione di

Quindi oggi era che di astrazione in cui si va specializzando tipo facendo di fare un bel tempio rispetto al padre.. Insieme l'ereditarietà sia polimorfismo degli operatori che permette di definire la stessa operazione con lo stesso nome sia oggetti diversi l'operazione fa la stessa cosa di lizzano metodi diversi a seconda dell'oggetto c'è provo con corona semplificazione del disegno.. Le due cose messe assieme provo con una disciplina di disegno particolare che ha degli impulsi notevole sull'integrazione uomo macchina.

**Impedence Mismatch:** La separazione tradizionale tra linguaggi di programmazione e DB dovrebbe essere evitata. Quindi si dovrebbe usare unico ambiente con dati persistenti ossia eliminiamo la parte embedded sul BD e consideriamo il DB come avente dei dati con la differenza che i dati sono persistenti.

**Applicazioni con oggetti complessi:** il problema dell'applicazione di oggetti non tradizionali è che non sono gestiti direttamente dal sistema di BD e se vogliamo aumentare le capacità di sistema di gestire oggetti che non sono ADT e di gestire oggetti effettivamente complessi.

**Performance:** se i sistemi di DB relazionali sono lenti mai il programmatore astuto se usa un ambiente unico con dati persistenti può programmare l'accesso ai dati maniera efficace.

Alcuni problemi di questo approccio uniforme sono:

1) **incapsulamento** in sistemi di DB il fatto che l'unico modo per accedere dati sia attraverso le funzioni ci impedisce di fare una la serie di cose.

2 **associazioni:** l'approccio OO lavora sul modello funzionale in cui associazioni vengono rappresentate come attributi di oggetti (analogamente alla presentazione che usiamo in DB relazionali).

Nel modello tabellare relazionale faccio ciò maniera molto agevole in quanto con le tabelle definiamo in maniera molto agevole sia la funzione principale del suo inverso. Il passaggio dal modello funzionale complica ulteriormente la vita sia perchè per ogni funzione bisogna definire inversa sia perchè comincia a sparpagliare i dati mantenendo sempre un record ma perdendo l'insieme dei record quindi le formazioni dell'associazione diventano parte dell'oggetto stesso.

**Concetti fondamentali di OODBMS:** standard ODMG per OOBDMs.

**Concetto di identità dell'oggetto:**

- Identificativo unico e immutabile
- Indipendente delle altre caratteristiche dell'oggetto
- Non è solitamente un indirizzo fisico.

Esiste un tecnica **POINTER SWIZZLING** (scambio di puntatori) che una volta che essa usa l'indirizzo fisico dell'oggetto sostituisce o copia l'indirizzo fisico in quello logico. Da quel

momento in poi non abbiamo più bisogno della Tavola di Traduzione ma accediamo direttamente all'indirizzo fisico.

C'è una divisione tra oggetti e valori letterale

Oggetti: sono tipicamente complessi, possono essere strutturati o non strutturati. Gli oggetti inoltre hanno identificazione tramite OID.

Valori letterali: Non hanno OID e hanno solo il valore.

Gli oggetti sono costituiti da altri oggetti o da valori, tramite l'utilizzo di costruttori di tipi.

I tipi utilizzati per costruire gli oggetti sono predefiniti:

- Set (insieme)
- List (insieme ordinato)
- Bag (multinsieme)
- Array (vettore)

Tutti questi sono collezioni di OID, permettono di mantenere insieme gli oggetti in set, list...

Un altro tipo sono le tuple, ovvero un aggregato di OID.

Esempio

Ox(OID, COSRTUTTORE, VALORE)

Nel caso in cui i valori sia dello stesso tipo posso costruire dei set in questo modo:

O7(I7, SET, {i1,i2,i3})

In questo modo si definisce un oggetto come una struttura complessa partendo da Costruttori di oggetti. Tutti questi oggetti, anche quelli complessi, sono identificati da un ODI e sono indirizzabili da altri oggetti.

Ho così definito una struttura complessa a grafo che parte da (ad esempio) o7.

Uguaglianza tra due oggetti:

In questo contesto diventa complesso il concetto di uguaglianza tra due oggetti. Dati due oggetti si possono avere due tipi di uguaglianza:

- l'uguaglianza profonda in cui si richiede la stessa struttura gli stessi oggetti allo stesso punto della struttura e quindi gli stessi valori inoltre ci sarà lo stesso stato
- l'uguaglianza superficiale rilassa il tipo di uguaglianza precedente concentrandosi sullo stato, sugli stessi valori e OID diversi.

**Incapsulamento:** Quello che implica dal definizione di ADT è un incapsulamento totale in cui tutte le proprietà sono invisibili e direttamente accessibili attraverso operazioni definite all'interno dell'ADT stesso.

Dal punto di vista di sistemi dei DB è altamente inefficiente perché abbiamo bisogno di un accesso diretto agli attributi definite all'interno della struttura per permettere di avere un accesso ai dati più veloce.

Questo approccio viene quindi rilassato facendo una separazione tra gli attributi che possono essere in parte visibile e in parti nascosti.

Stato: gli attributi possono essere:

- Visibili : tipicamente in lettura e gestiti in scrittura dai metodi.
- Nascosti

Operazioni

- Interfaccia/Signatre: Prototipo della funzione
- Metodo/implementazione: Implementazione della dichiarazione della funzione

Il modo per definire l'incapsulamento è la definizione di classe. Si utilizza la dot notation per accedere alle variabili o operazioni di uno specifico oggetto.

## Concetto di Persistenza

Non tutti gli oggetti che servono per la logica del programma sono persistenti: sono definiti degli oggetti temporanei.

Oggetti persistenti: rendiamo persistenti gli oggetti attraverso due 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 numero di oggetti ma non possiamo dare un nome a tutti gli oggetti che voglio rendere persistenti.
- **2) Per Raggiungibilità:** B è raggiungibile da A se esiste una sequenza di riferimenti che da A porta a B. Tale concetto è usato nel seguente modo:  
*se A è persistente allora ogni B raggiungibile da A è persistente*

Con 1 e 2 messi insieme mi permettono di avere una certa flessibilità:

*Posso definire con un nome un oggetto persistente in modo che tutto ciò che è raggiungibile da lui diventa persistente.*

In questo modo stiamo quindi simulando delle Relazioni. Questo meccanismo che permette di raggruppare logicamente gli oggetti viene chiamato EXTENT e fa parte dello standard ODMG.

## STANDARD ODMG 2.0

Questo standard è composto da 4 parti:

- Modello ad Oggetti
- Linguaggio di Definizione (ODL)
- Linguaggio di Interrogazione (OQL)
- Binding a linguaggi di Programmazione (OO)

## Modello ad Oggetti

Abbiamo una distinzione tra:

**Oggetti** che sono destritti da:

OID: unico nel sistema

Tempo di Vita: Persistente o Transitorio

Struttura: Collezione o Atomico

**Valori Letterali** che sono parte dell'oggetto e non hanno un OID possono essere:

Valori Atomici (long, string..)

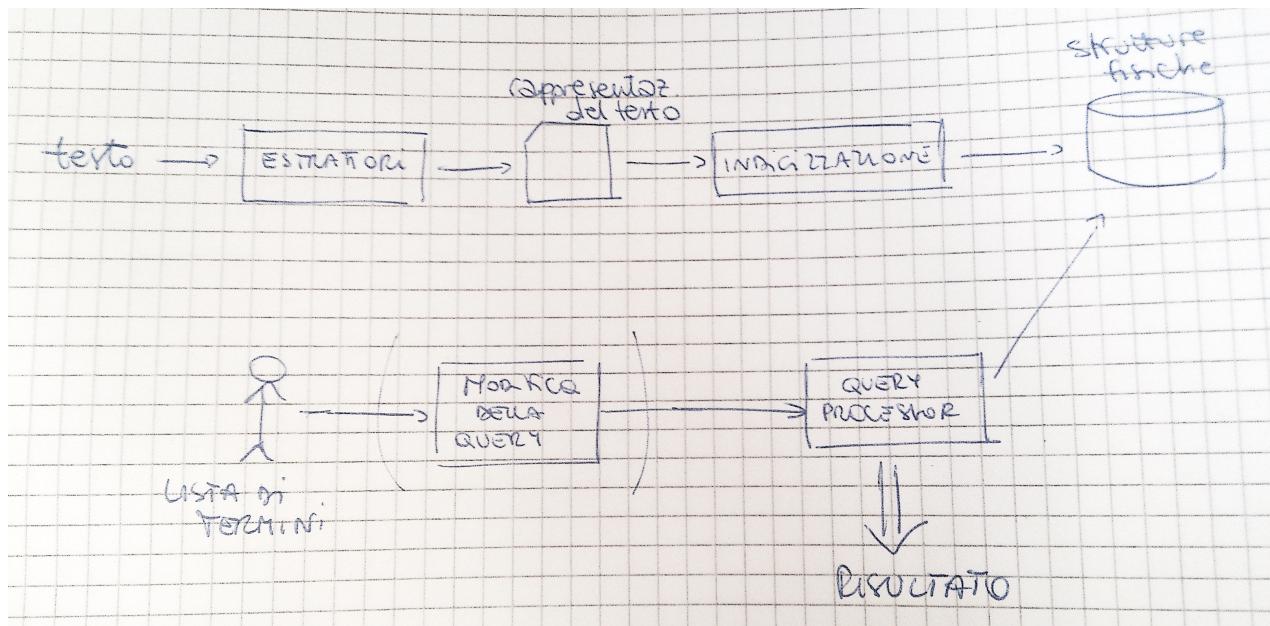
Strutturati (Struct)

Collezioni ( Oggetti, Valorui)

## Information retrieval, text retrieval e motori di ricerca

Lo schema di base dell'indicizzazione è il seguente: abbiamo un testo che viene dato in input ad un estrattore che produce un rappresentativo del testo che contiene una lista di termini su cui lavoreremo e rappresenta il testo iniziale. A questo punto il testo iniziale non verrà più utilizzato. La lista di termini viene passata ad un indicizzatore che alimenta delle strutture interne di indici (che sono strutture fisiche) che permettono un accesso rapido al documento.

Dall'altra parte abbiamo la parte di query dove l'utente inserisce una lista di termini in un query language di qualche tipo. La query viene, eventualmente, passata ad una componente che si occupa di modificarla (ad esempio gestendo i sinonimi, etc.). La query (modificata o non) viene passata al processor che accede alla struttura fisica per recuperare i dati.



Le strutture fisiche sono generalmente gestite come liste invertite. Le liste invertite hanno una chiave associata ad una lista contenente i puntatori agli oggetti che sono associati a quella chiave. Accanto ad una struttura di questo tipo deve esserci un repository che si occupa di mappare l'id del documento con il documento effettivo (o la sua posizione).

Recuperiamo i testi sulla base di stringhe di caratteri. Il livello semantico rispetto cui ci muoviamo è molto basso: se cerco cane e il documento contiene "cani" non viene trovato. Vi è un enorme gap semantico tra utente e sistema. Il sistema si muove sulla base di stringhe mentre l'utente si muove sulla base di concetti. Questo causa numerosi problemi:

- 1) Varianti sintattiche e inflessionali (cane e cani ad esempio)
- 2) Sinonimi e concetti semanticamente collegati
- 3) La tematica del documento è ignorata o il termine specifico non appare all'interno del documento

Il **recall** è l'esaurienza ed è la capacità di recuperare tutti i documenti rilevanti.

La **precision** o assenza di rumore è la capacità di recuperare solo i documenti rilevanti.

Chiamiamo:

- R, documenti rilevanti
- A, insieme dei documenti dati come risposta alla query
- I, R intersezione A, ovvero i documenti rilevanti recuperati dalla query

$$\text{Recall} = I / R$$

$$\text{Precision} = I / A$$

## Termini

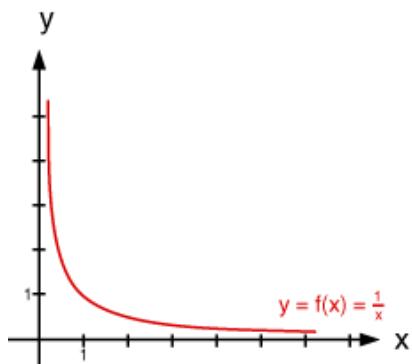
All'inizio non era utile indicizzare l'intero contenuto del documento, ma si indicizzava un insieme chiamato **white list** che era un vocabolario controllato di termini rilevanti. Abbiamo anche una **black list** o stopwords che contiene dei termini che vengono completamente ignorati in fase di indicizzazione (ad esempio gli articoli, le voci dei verbi servili). Con le stopwords otteniamo un risparmio in termini di risorse interne.

La distribuzione delle parole segue una legge nota come legge di **zipf** definita come:

$$p_i = C / i$$

$$\text{la sommatoria di tutte le } p_i \text{ deve essere } = 1 \text{ e } C = 1 + \frac{1}{2} + \dots + \frac{1}{n} = \Sigma_k$$

È una legge  $1/x$  perché il suo grafico assume la forma seguente:



Esiste un numero piccolo di parole che compaiono molto spesso e una lunga coda di parole molto rare.

## Problemi

1. **Tokenizzazione:** Come tokenizziamo? In base alla virgola, lo spazio, etc. Alcuni termini non sono comunque tokenizzabili perché se contiamo solo caratteri alfanumerici, la parola 'C++' non viene tokenizzata, ma viene tokenizzato solo 'C'. Questo comporta una perdita di recall.
2. **Stopwords:** alcune stopwords possono causare una notevole perdita di precision. Se considero 'di' come stopword, non potrò cercare 'di Pietro' ma solo 'Pietro'.
3. **Inflessioni:**
  - a. Riduzione a tema della parola (stemming). Anziché ricordare 'cane' e 'cani' ricordo solo 'can'. Quando l'utente cerca 'cane' il sistema fa una ricerca sul tema ed effettua la query su 'can'. Non necessariamente il tema rappresenta univocamente una serie di inflessioni che hanno lo stesso significato. Consideriamo il tema 'neutr' che è il tema di 'neutro', 'neutrale', 'neutralità', '**neutron**'. Se cerco uno dei primi termini può andare bene la riduzione a tema, ma se cerco neutrone la precision si riduce di molto in quanto non sarò interessato alle ricerche relative alle prime tre parole. Un altro problema sono le forme irregolari, ad esempio i verbi irregolari spesso non hanno lo stesso tema (es. verbo andare). Inoltre trovare il tema di una parola non è sempre facile. Questo fa sì che spesso la riduzione a tema consiste nel prendere i primi n caratteri del termine.
  - b. In alternativa alla riduzione a tema possiamo utilizzare un qualche tipo di normalizzazione grammaticale. Invece di cercare un tema rappresentativo di una classe di termini con desinenze diverse cerchiamo una forma grammaticale normale: ad esempio il femminile plurale per gli aggettivi, l'infinito per i verbi,

etc. Dal punto di vista grammaticale facciamo sempre una riduzione all'infinito dei verbi e consideriamo sempre solo la forma normale sia per l'indicizzazione che per la ricerca. Questo riduce le strutture interne e aumenta l'efficienza della ricerca. Ci sono delle situazioni in cui una riduzione di questo tipo aumenta il rumore. Supponiamo di avere 'uomo' a seconda della convenzione adottata può essere tradotto in 'donna' anche se non è uguale! Abbiamo di nuovo aumentato il recall e diminuito la precision.

4. Sinonimi e termini correlati vengono solitamente gestiti da un meccanismo chiamato thesaurus che identifica il dizionario dei sinonimi. Ci interessa la sinonimia ma ci interessano molto di più i termini correlati. All'inizio i thesauri erano realizzati come liste di sinonimi (o termini correlati), adesso invece sono organizzati come reti semantiche. Capito il contesto dell'interrogazione è possibile quindi migliorare notevolmente la ricerca utilizzando i sinonimi e i termini correlati.
5. Termini rilevanti non sono presenti, per questo vengono utilizzate delle white lists con keywords

In conclusione il **termine** è quello che indicizziamo e che, insieme ad altri, rappresenta il documento. Non necessariamente tutti i token del documento compaiono in questo insieme di termini rappresentativi e l'insieme originale può essere ampliato ad esempio con sinonimi.

Per permettere delle ricerche con virgolette "parola1 parola2" abbiamo bisogno di memorizzare negli indici anche la posizione della parola all'interno del documento (anzi una lista di posizioni dove appare il termine nel documento). La dimensione degli indici esplode!

I primi sistemi commerciali (il più famoso è IBM stairs) avevano degli indici con dimensione 3 volte tanto il corpus iniziale. Adesso abbiamo la possibilità di avere indici con un overhead del 50% del corpus.

Quello che vogliamo fare è fare un tipo di ranking del nostro risultato. Robertson e Sparck-jones hanno creato un relevance ranking principle che serve per ordinare i risultati per rilevanza decrescente. Con questi meccanismi aumentiamo la precision mantenendo inalterata l'esaurienza perché i documenti più precisi vengono visualizzare prima. Possiamo utilizzare dei pesi per caratterizzare i termini, utilizzando dei sistemi di weighting che stimano la rilevanza dei termini in base all'interrogazione. Questi si basano solitamente sulla frequenza dei termini nel documento.

Il meccanismo usa una fuzzy logic che caratterizza l'appartenenza di un elemento a un insieme è caratterizzato da una probabilità e non da un valore true o false come nella logica booleana. Maggiore è la probabilità di appartenenza di un termine e più è rilevante il documento.

Indichiamo con  $p_e(A)$  la probabilità che l'elemento appartenga all'insieme A.

- $p(A \text{ and } B) = \min(p(A), p(B))$  - intersezione
- $p(A \text{ or } B) = \max(p(A), p(B))$  - unione
- $p(\neg A) = 1 - p(A)$

Chiamiamo T e T1 dei termini e Ft(doc) la frequenza del termine t nel documento 'doc'. Utilizzando il meccanismo della logica fuzzy allora abbiamo che:

Query	Rilevanza
T	Ft(doc)
T and t1	Min(ft(doc), ft1(doc))
T or t1	Max(ft(doc), ft1(doc))

T - t1 **	Ft(doc) - ft1(doc)
-----------	--------------------

Dove \*\* è t and not t1. Il numero di occorrenze di t1 riduce l'importanza del documento.

### TF – IDF

Come arriviamo ad un peso che tenga conto non solo della frequenza del termine nel documento ma considerando anche la dispersione del termine sulla banca dati.

Tf indica Term Frequency, IDF indice Inverse Document Frequency che indica la specificità del termine nel documento.

Il problema di una stima TF che utilizza direttamente la frequenza del termine per calcolare la rilevanza è che tende a favorire i documenti dove il termine è ripetuto molte volte indipendentemente dalla loro lunghezza. Un documento di 100 righe dove 'cane' appare due volte dovrebbe essere più rilevante rispetto ad un documento di 1000 righe dove 'cane' appare 3 volte.

Per questo motivo solitamente si utilizza la frequenza pesata in maniera logaritmica:

- $1 + \lg_2(f_t(d))$  se la frequenza è > 0.

In questo modo il peso non è più direttamente proporzionale alla frequenza ma è logaritmicamente proporzionale (le prime occorrenze sono molto rilevanti e via via che aumentano diventano meno rilevanti).

Con IDF l'idea è di misurare la specificità del termine utilizzando ZIPF. I termini molto frequenti o molto rari sono poco interessanti, mentre quelli nel mezzo hanno maggiore rilevanza.

Utilizzando solo TF ad esempio gli articoli avrebbero dei pesi elevatissimi! Vogliamo integrare TF con IDF per favorire dei termini più generali e non troppo frequenti.

L'idea è di prendere il vocabolario, ordinare i termini in maniera decrescente per frequenza di documenti (rank) e caratterizzare la curva corrispondente. I primi saranno le stopwords mentre la coda saranno i termini che appaiono molto raramente nel corpus.

A seconda del rank del termine abbiamo la frequenza dei documenti che è dato da

$$- n(r) = C * r^{(-\alpha)}$$

dove C è una costante di normalizzazione che tiene conto della dimensione del corpus e alpha si chiama costante empirica.

Se poniamo alpha = 1 che è un'approssimazione ragionevole per i testi in inglese, abbiamo che il log n(r) è uguale a  $\log(C) - \log(r)$ .

Log n(r) = lg C - lg r, se normalizziamo otteniamo che  $\lg r \approx \lg N - \lg(n(r))$ , definiamo  $\lg r$  come  $idf_i = \log(N / n_i)$ . Lo schema totale è dato da Tf \* Idf. Per le stopwords idf tende a 0, mentre per i termini poco frequenti tende alla dimensione del corpus.

TF\*IDF ha un grafico del tipo:

?? img grafico 13/11

una possibilità è di usare  $TF*IDF = (1 + \log(f(i,j))) * \lg(N/n_i)$

?? rivedere la parte di calcoli relativa a idf

I modelli di tipo fuzzy sono relativamente recenti. In precedenza era stato definito un modello vettoriale (che viene utilizzato tuttora).

### Modello vettoriale (SMART)

Ha un'interpretazione geometrica molto precisa. L'idea è di definire uno spazio vettoriale a T dimensioni, dove T è il numero di termini. Ogni dimensione corrisponde ad uno specifico termine. Lo spazio vettoriale viene utilizzato per caratterizzare sia i documenti che le queries. La nostra rappresentazione del documento è un vettore  $D(w_0, \dots, w_t)$  dove  $w_i = 0$  se il termine non appartiene al documento oppure  $= tf * idf$  se appartiene al documento. La query è rappresentata esattamente allo stesso modo. La query è una lista di termini ad ognuno dei quali è possibile assegnare un peso diverso,  $Q(w_0, \dots, w_t)$ .

Il vocabolario contiene spesso milioni di termini e quindi lo spazio vettoriale risulta molto elevato! Il vantaggio è la possibilità di definire matematicamente una misura di similarità tra documento e query grazie all'omogeneità nella rappresentazione delle due entità.

La somiglianza tra X e Y è data dal prodotto scalare di X con Y diviso la norma.

$$S(X, Y) = (X^*Y) / (||X|| * ||Y||) = \text{sommatoria su } i \text{ di } (x_i * y_i) / [\sqrt{\text{(sommatoria su } i \text{ di } x_i^2)} * \sqrt{\text{(sommatoria su } i \text{ di } y_i^2)}]$$

Un altro coefficiente di somiglianza è dato dal coefficiente di Jaccard:

$$S(X, Y) = |X \text{ intersez } Y| / |X \cup Y|$$

Rappresentando documenti e query in questo modo possiamo calcolare una somiglianza tra due documenti. Non abbiamo una logica booleana. Il problema principale in ogni caso è la dimensione dello spazio vettoriale. Per risolvere questo problema si usano diverse tecniche:

- **clusterizzazione:** la tecnica del clustering ovvero di raggruppamento di documenti simili in modo da evitare il confronto tra query e tutti i documenti sostituendolo con il confronto tra query e rappresentante di un cluster.  
Documenti simili vengono reperiti insieme. Se faccio un'interrogazione e recupero un documento -> tutti i documenti simili saranno rilevanti. Seguendo questa intuizione raggruppo i documenti in gruppi (clusters) in cui la somiglianza fra i documenti sia più forte rispetto alla somiglianza con i documenti di altri gruppi. Sceglio un rappresentativo di ogni gruppo e lo chiamiamo "centroide" ovvero il baricentro del cluster. La clusterizzazione è una tecnica di classificazione dei documenti.  
Il BM25 è un famoso algoritmo di ranking e si dimostra che ha una precisione maggiore del modello vettoriale. ??integrare-bm25.  
Mentre per il recall abbiamo bisogno di un esperto, per la precisione abbiamo solo bisogno di vedere la precisione della query in punti specifici (solitamente 5, 10, 20). Per confrontare gli algoritmi viene calcolata la precisione.
- Una seconda tecnica è di filtrare i documenti evitando i confronti con documenti che non contengono nemmeno uno dei termini della query.

### Search engine architecture

?? vedere parte slide dei motori di ricerca

Ci sono sostanzialmente due filoni in questo campo:

- fornire suggerimenti all'utente per rendere l'interrogazione migliore, proponendo query più precise o rilevanti rispetto a quello che cerca l'utente. Spesso sono previste funzioni di correzione ortografica.
- L'altro filone, rappresentato dal onebox di Google, che interpreta la ricerca non come il reperimento di documenti rilevanti ma come la risposta ad una domanda come "Chi è il governatore della California?". Oppure richiesta di conversioni, calcoli, previsioni del tempo, etc.

## Tassonomie dinamiche