SISTEMI INFORMATICI E SISTEMI INFORMATIVI (cap.1 del libro)

Argomento principale di questo anno scolastico sarà la generazione, raccolta, archiviazione elaborazione e trasmissione di DATI. Queste attività rappresentano uno dei rami più importanti e diffusi di applicazione dell'informatica (ricordo la macchina tabulatrice di Hollerit, ideata per raccogliere i dati del censimento USA del 1890, https://www.youtube.com/watch?v=6TBfJBBN6Zs).

DATI E INFORMAZIONE

E' dal primo anno di questa scuola che sottolineiamo la differenza fra questi due concetti. Facciamolo ancora una volta:

- Dato: **misura** di un **fenomeno** che descrive un aspetto della realtà che siamo interessati a rappresentare.
- Informazione: **incremento della conoscenza** del fenomeno che stiamo osservando ottenuta grazie alla **elaborazione** dei dati.

Esempio:

Un medico, utilizzando un termometro misura la temperatura corporea di una persona e rileva che essa è di 39°C. La temperatura corporea è un dato.

Il medico, confrontando il dato con la temperatura media delle persone sane che è di circa 36°C (elaborazione del dato) determina che il paziente è malato (il paziente ha la febbre è una informazione).

Spesso la raccolta di più dati relativi al fenomeno osservato consente di ottenere informazioni più precise e quindi consente di comprendere meglio il fenomeno osservato. Al contrario invece, alcuni dati possono essere non utili o addirittura fuorvianti nella comprensione del fenomeno. Tali dati vanno scartati.

Esempio:

Con le analisi del sangue (valori dei globuli rossi, bianchi, ecc..) il medico può fare ipotesi più precise sulla patologia che affligge il paziente con la febbre. (Questo è il caso in cui più dati consentono una migliore comprensione del fenomeno).

Dati inutili o fuorvianti: ad esempio se il paziente porta i suoi esami del sangue di 10 anni fa, i dati sono fuorvianti.

Esempio dato fuorviante. Attenzione in Trentino in due anni gli omicidi sono aumentati del 200%! E' pericolosissimo andare in Trentino!

pubblicato dall'Eures, l'Istituto di Ricerche economiche e sociali. Analizzando il rapporto del Viminale, relativo agli anni **2014/2016, nelle Regioni dove c'è stato un aumento di omicidi, la percentuale è quasi completamente assorbita proprio dai delitti commessi in famiglia**. Il dato del Trentino per esempio è impressionante: +200%. Se si guardano i numeri, però, si scopre che si è passati da 1 omicidio nel 2014 a 3 del 2016, e i 2 morti in più non sono imputabili a un fatto di ordinaria criminalità (e quindi ad una mancanza di sicurezza), ma ad un padre impazzito che ha ucciso la moglie e il figlio. Lo stesso discorso vale per l'Abruzzo (+50%), per

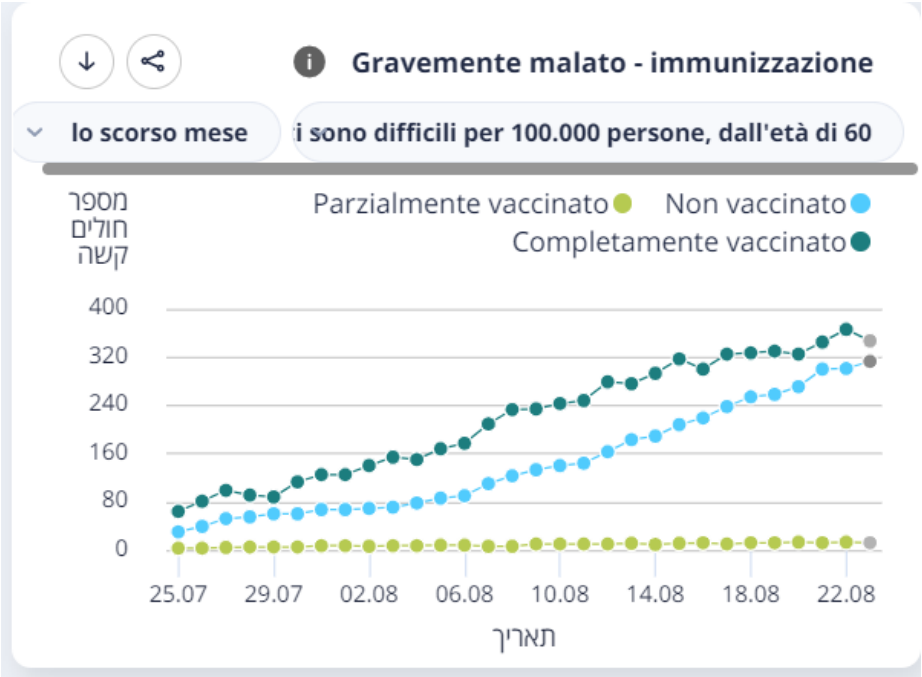
Faccio un altro esempio in cui scegliendo quali dati mostrare posso creare un sentimento piuttosto che un altro nell'opinione pubblica: se voglio diffondere il concetto che i vaccini contro il COVID non funzionano, mostro questo:

Dati sui ricoveri in condizioni gravi per Covid 19 in Israele, over 60, valori assoluti. Prendiamo il giorno 12/8/2021

Completamente vaccinati: 248

Non vaccinati: 144

Parzialmente vaccinati: 10



Deduzione non corretta: i vaccini non funzionano. Sono più numerosi i ricoverati vaccinati di quelli non vaccinati.

Ma se guardiamo il grafico con il numero di ricoverati OGNI 100 000 abitanti nelle tre popolazioni (vaccinati, parzialmente vaccinati, non vaccinati) si capisce di piu'. Dati dello stesso giorno:

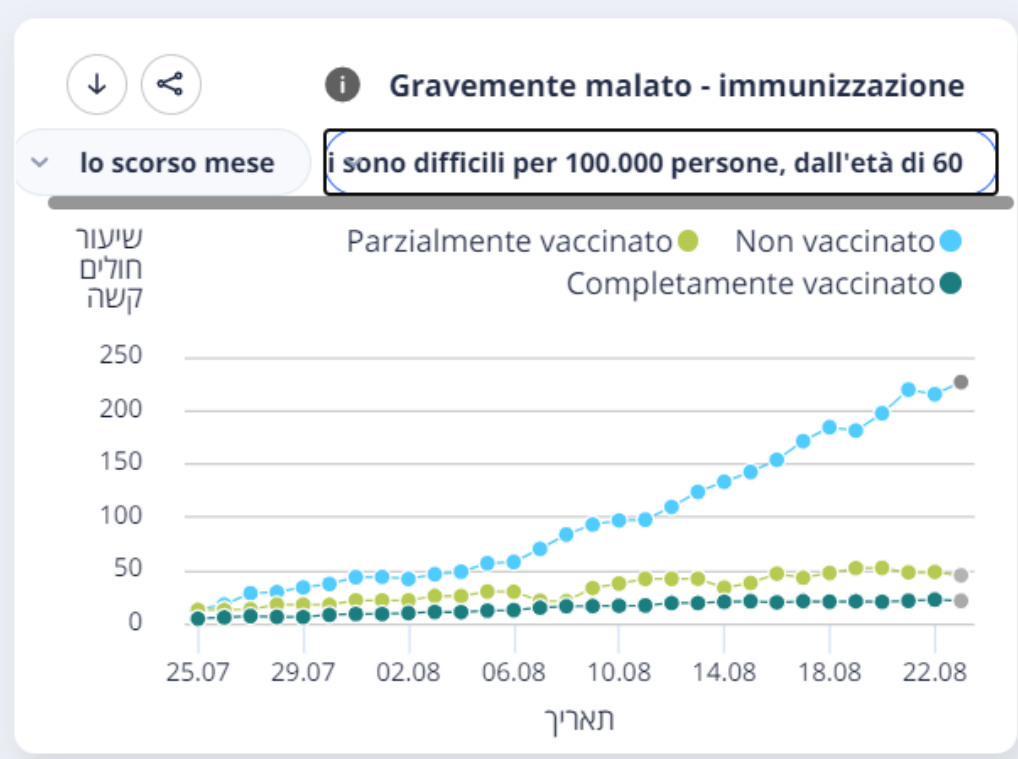
giorno 12/8/2021

Completamente vaccinati: 19.1 ogni 100000 abitanti

Non vaccinati: 109.5 ogni 100000 abitanti

Parzialmente vaccinati: 41.9 ogni 100000 abitanti

Poiché le restrizioni (distanziamento, mascherina ecc..) sono le stesse per tutti, il numero di malati gravi ha un'incidenza 5 volte maggiore fra i non vaccinati rispetto ai vaccinati. Deduzione: i vaccini riducono, pur non portandola a 0, la probabilità di contrarre gravemente la malattia.



Quindi facciamo attenzione a come ci presentano i dati! Sempre interpretarli con attenzione. I dati sono una cosa, le informazioni sono un'altra cosa.

E' importante osservare che le informazioni hanno spesso una durata limitata nel tempo, per questo **il tempo** è un aspetto fondamentale nel processo di comprensione di un fenomeno finalizzato a risolvere problemi.

Esempio:

Il medico è molto bravi a raccogliere i dati, scartare quelli inutili, elaborarli correttamente… trovare la diagnosi……ma il paziente muore! ……Troppo tardi!

SISTEMI INFORMATICI E SISTEMI INFORMATIVI

Ma perché è utile ottenere informazioni per la conoscenza dei fenomeni?

Le motivazioni si possono raggruppare in due scopi:

- **scopo operativo**: un generico ente (ente pubblico, società, organizzazione) ha lo scopo di raccogliere dati ed ottenere informazioni per le proprie attività operative

  Esempio: l'amministrazione comunale necessita dei dati dei propri cittadini per rilasciare certificati di residenza, carte d'identità ecc..

- **scopo decisionale (di governo)**: un generico ente (ente pubblico, società, organizzazione) ha lo scopo di raccogliere dati ed ottenere informazioni per poter prendere delle decisioni sulla programmazione delle proprie attività

  Esempio: un supermercato raccoglie dei dati per determinare quali prodotti sono più venduti, o quali prodotti danno un guadagno maggiore e fare opportune scelte sulla politica di acquisto (non vendo più un certo prodotto….) o di marketing (faccio l'offerta su quel prodotto…)

Alcune informazioni possono essere sia operative che di governo. Ad esempio i costi relativi agli stipendi del personale di una azienda danno sia informazioni operative (necessarie per la produzione delle buste paga dei dipendenti) che decisionali (è possibile/opportuno assumere altre persone?)

Alla luce di quanto detto definiamo quindi cosa è un sistema informativo.

Con il termine **sistema informativo (S.I.)** si indica l'insieme delle **procedure** e delle **risorse** (materiali e umane) finalizzate alla

- raccolta
- archiviazione
- elaborazione
- trasmissione

di **dati** per ottenere informazioni sia per scopi operativi che decisionali di un'organizzazione.

Attenzione: il sistema informativo non è necessariamente legato all'informatica. Tali sistemi infatti

1. esistevano prima dell'informatica (banche, comuni hanno sempre gestito dati…)
2. sono formati dalle procedure, dalle risorse umane e dalle risorse materiali, ed è quest'ultima l'unica parte in cui ha un ruolo l'informatica.

Quindi un S.I. è indipendente dagli strumenti utilizzati per la gestione dei dati e delle informazioni.

Allora cosa c'entra l'informatica?

L'informatica ha consentito di **automatizzare** la raccolta, archiviazione, elaborazione, trasmissione di dati rendendo queste operazioni molto più veloci (ricordiamo l'importanza del fattore tempo).

Definiamo quindi cosa si intende quando si parla di sistema informatico.

Con il termine **sistema informatico** si intende una componente (una parte) di un sistema informativo, e precisamente la componente che consente di **automatizzare** (rendere automatiche) le operazioni di raccolta, archiviazione, elaborazione, trasmissione dei dati in forma digitale attraverso dispositivi informatici (hw e sw).

Si pensi a quanto è aumentata l'efficienza di queste operazioni con i sistemi informatici. Ad esempio: aggiornare la planimetria di un edificio nel sistema informativo del catasto edifici, prima dell' informatizzazione del sistema, richiedeva molto più tempo. Altro esempio: Lo studente non può più bruciare a scuola , l'informatizzazione ha reso il sistema informativo molto più efficiente che vi piaccia o no.

Altro esempio: come si faceva ad ottenere il certificato di residenza fino al 15/11/2021? Come si fa ora? Vedi qui: https://www.anagrafenazionale.interno.it/

CICLO DI VITA DI UN SISTEMA INFORMATICO

Come abbiamo detto un sistema informatico è una parte del sistema informativo, uno degli errori più comuni quando si progetta un sistema informatico è quello di immaginarlo come una semplice automatizzazione delle procedure eseguite manualmente. In realtà la realizzazione di un sistema informatico all'interno di un sistema informativo è un'attività complessa che deve portare ad una razionalizzazione delle attività svolte al fine di migliorare l'efficienza di un sistema informativo.

Già l'anno scorso in TPS avevamo visto il ciclo di vita del software. In particolare avevamo visto diversi modelli (modello ciclico, modello a spirale) e avevamo accennato ad un particolare approccio di metodologie di progettazione (metodologie AGILE). Sostanzialmente per progettare il software non esiste una sola metodologia, ne esistono diverse, ciascuna più o meno formalizzata.

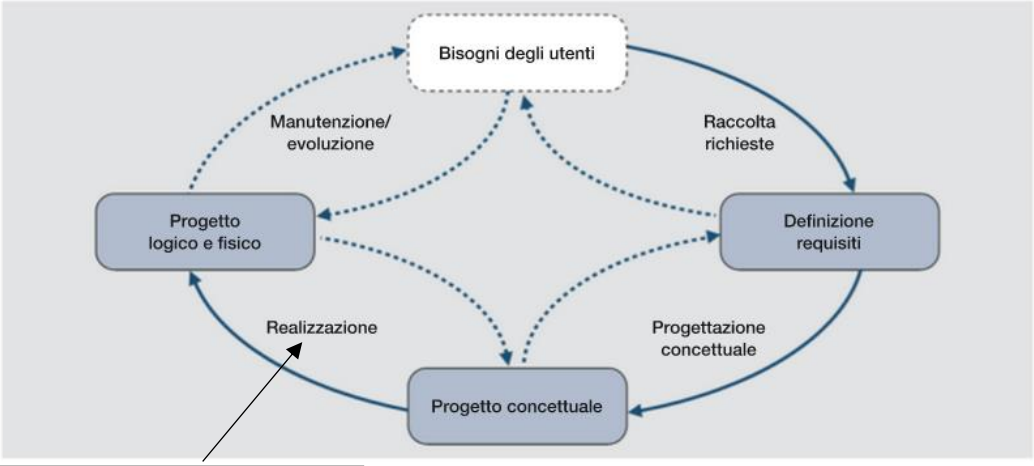
Ciò che è sempre vero, qualunque metodologia venga utilizzata, è che la realizzazione di un software è caratterizzata da operazioni che si ripetono ciclicamente in cui l'output di una fase diventa l'input della fase successiva.

Nelle varie rappresentazioni del ciclo del ciclo di vita che vedemmo l'anno scorso non si parlò mai della memorizzazione dei dati e della loro struttura. L'unico modo che abbiamo visto per memorizzare i dati è quello di salvare i dati su file binari o su file di testo. Questa modalità va bene finchè i dati da memorizzare sono pochi, ma per sistemi informatici più complessi, in cui i dati da memorizzare sono molti si utilizza uno strumento software chiamato database, e la progettazione delle applicazioni inizia proprio dalla modellazione dei dati prima ancora di individuare classi, metodi ecc.

Quest'anno, come vedremo, ci occuperemo prevalentemente delle tecniche e degli strumenti per la raccolta e memorizzazione dei dati.

Il ciclo di un sistema informatico, Come già visto l'anno scorso prevede delle fasi che vengono ciclicamente ripetute in cui l'output di una fase diviene l'input della fase successiva:

1. **raccolta delle richieste degli utenti**, che produce come output la **definizione dei requisiti** a cui il sistema informatico dovrà esser conforme a partire dai bisogni dell'utente.
2. La **progettazione concettuale**, che produce come output il **progetto concettuale.**
3. La **realizzazione** (o **progettazione logica** e **fisica)**, che produce come output **l'insieme delle componenti hardware e software che implementano il sistema informatico.**
4. **La manutenzione/evoluzione del sistema software**



Comprende l'implementazione e il test.

Nei piccoli progetti che abbiamo realizzato l'anno scorso con i file, abbiamo visto che un qualunque sistema informatico richiede che i dati vengano memorizzati (nel nostro caso su file di testo o su file binari).

Nel corso di questo anno scolastico studieremo principalmente le componenti software che permettono di memorizzare grandi quantità di dati in maniera più efficiente di quanto si possa fare semplicemente con i file di testo o con i file binari, ossia con l'utilizzo di appositi sw chiamati DBMS (Data Base Management System).

La ciclicità delle fasi è dovuta, oltre che a eventuali richieste correttive sul sistema informatico da parte dell'utente, al fatto che la realtà che si vuole rappresentare con il sistema informatico è sempre in evoluzione, sia per motivi esterni (ad esempio nascita della rete internet, introduzione di nuove tecnologie…) sia per motivi interni (espansione o modifica delle attività dell'organizzazione che utilizza il sistema informatico). Tali modifiche costringeranno quindi ciclicamente a modificare il sistema informatico.

Approfondiamo le prime 3 fasi per cercare di capire meglio in cosa consistono.

Fase 1: raccolta delle richieste degli utenti

Suddividiamo la fase 1 in 3 sottofasi.

- Fase 1 A: indagine preliminare relativa all'introduzione del sistema informatico nell'organizzazione.

  Questa fase viene generalmente svolta dall'organizzazione che intende informatizzare il proprio SI. L'informatizzazione di un SI porta senz'altro dei vantaggi a lungo termine ma è onerosa economicamente e crea inizialmente delle problematiche quindi l'organizzazione deve valutare se la realizzazione del sistema informatico è "conveniente".

  1. Individua in quali settori dell'organizzazione potrebbe essere conveniente introdurre il sistema informatico (magazzino? produzione? contabilità? tutti?)

2. valuta l'impatto dell'introduzione del sistema informatico sull'organizzazione. Ad esempio sarà necessario riqualificare il personale, questo è un problema spesso sottovalutato. ad esempio, negli enti pubblici (personale poco avvezzo all'utilizzo di dispositivi tecnologici)
3. valuta le risorse economiche disponibili e i costi per l'introduzione del sistema informatico
4. decide se e in quali settori realizzare il sistema informatico

- Fase 1 B: analisi del sistema informativo esistente (arriva l'analista!).

  Una volta che l'organizzazione ha deciso di realizzare il sistema informatico ecco che arriva l'analista. Si sottolinea che la realizzazione di un sistema informatico serve per automatizzare una parte del SI che già esiste nell'organizzazione. Per tale motivo è necessario comprendere il SI esistente. Come già visto l'anno scorso l'analista è un professionista informatico che si deve interfacciare con i vari elementi dell'organizzazione a diversi livelli (con i diversi esperti del dominio) per raccogliere e **formalizzare** (raccogliere in una forma che sia chiara a chi dovrà poi sviluppare il sistema informatico) i requisiti. L'analista deve comunicare con elementi dell'organizzazione a diversi livelli, dovrà infatti conoscere e comprendere i bisogni di tutti gli utenti che potrebbero essere molteplici e diversi (il direttore del supermercato ha bisogno di conoscere determinati dati per prendere decisioni sullo sviluppo aziendale, il magazziniere deve conoscere altri dati, ad esempio dove si trovano i prodotti, la cassiera altri dati, ad esempio il costo al kg di un prodotto ecc..). Il lavoro dell'analista è molto complesso, la competenza aumenta molto con l'esperienza.

- Fase 1 C: definizione dei requisiti

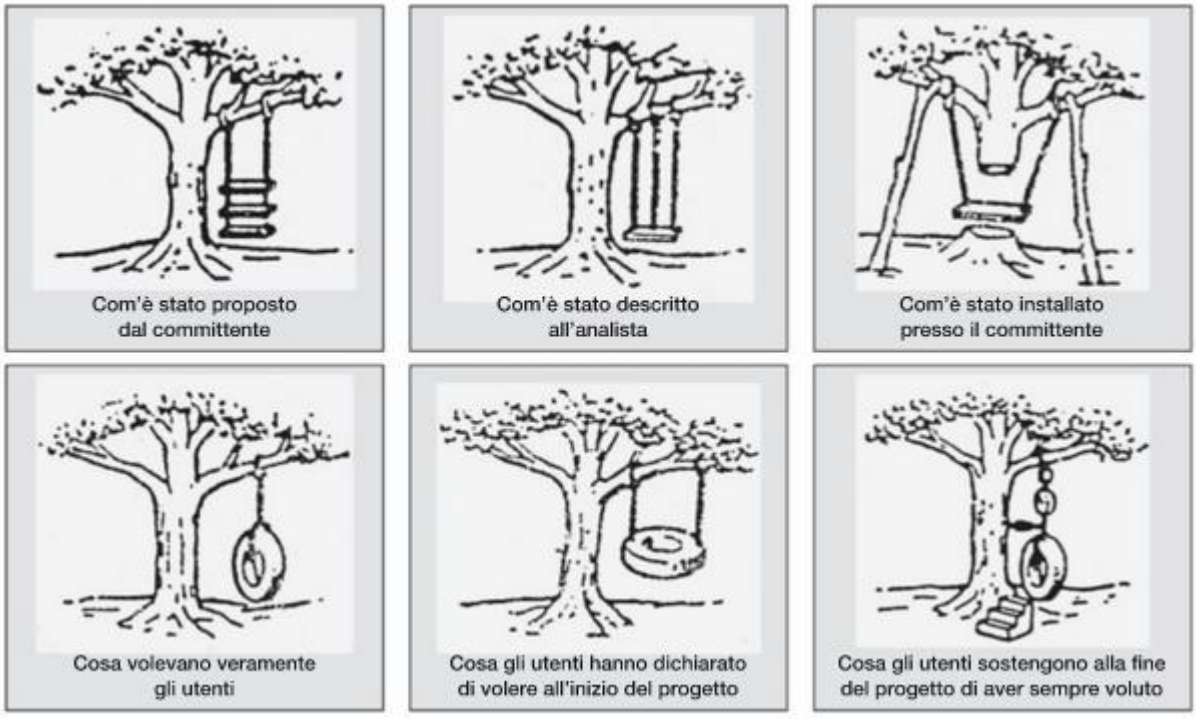
  Abbiamo già visto l'anno scorso le caratteristiche dei requisiti di un sistema software. Nella definizione dei requisiti relativi ai dati l'analista dovrà:

  - Classificare i dati: nome e tipo di ogni dato utilizzato
  - Indicare i **vincoli di integrità**, ossia le condizioni che devono rispettare i dati per essere **significativi e validi**. Ad esempio
    - La data deve essere in formato gg/mm/aaaa (oppure mm/gg/aaaa)
    - Non ci devono essere due fatture con lo stesso numero
    - ……
  - La descrizione delle procedure da automatizzare. Ad esempio:
    - Un prodotto del supermercato non può essere venduto se prima non è stato spostato da "magazzino" a "scaffale".
  - Il volume iniziale e la previsione di crescita dei dati nel tempo. Ad esempio:
    - Un supermercato ha 10 punti vendita ma bisogna prevedere la possibilità di aggiungere ulteriori punti vendita
  - Il **grado di privatezza** dei dati differenziato a seconda degli utenti e dell'utilizzo.

Questo punto è molto importante, infatti in una organizzazione complessa i dati sono moltissimi ed è quindi fondamentale che tutti gli utenti abbiano accesso solo alle informazioni che sono loro necessarie, questo per evitare:

1. Confusione dato da sovraccarico informativo (**information overloading**), ad esempio al dirigente del supermercato non interessa dove sono posizionati i prodotti nel magazzino.
2. Problemi di sicurezza dovuti ad informazioni riservate che solo alcuni utenti devono conoscere, ad esempio ogni cassiera deve conoscere solamente la propria password per accedere alla cassa del supermercato.

Nella seguente figura si rendono evidenti in maniera ironica le possibili difficoltà e incomprensioni fra analista, committente, utenti. Aspetto che già abbiamo potuto rilevare l'anno scorso nella realizzazione di progetti.



Fase 2: progettazione concettuale

In questa fase, la documentazione prodotta nella fase precedente viene utilizzata per realizzare un **modello astratto** del sistema informatico chiamato **progetto concettuale**. Il modello è definito "astratto" perché non specifica i dettagli implementativi, in particolare è **indipendente dalla tecnologia** hw e sw che verrà utilizzata in fase di realizzazione.

Il progetto concettuale consiste in una serie di documenti, schemi e diagrammi che descrivono:

- la struttura dei dati in termini di insiemi e relazioni fra essi
- i vincoli di ammissibilità dei dati

- l’integrità e la riservatezza delle informazioni

Nel nostro caso per rappresentare nel progetto concettuale la struttura dei dati e le loro interazioni, utilizzeremo il formalismo del diagramma Entità/Associazioni (diagramma E/R). Il diagramma ricorda alcuni dei diagrammi UML visti l’anno scorso, infatti il formalismo dei diagrammi UML ha ereditato alcuni concetti introdotti di diagrammi E/R.

In generale il progetto concettuale è interpretabile anche da non informatici, pertanto la sua realizzazione ha due scopi:

1. fornire un input per la fase successiva (progettazione logica e fisica)
2. consentire una verifica della correttezza dei requisiti individuati. Infatti il diagramma E/R è di facile lettura anche per gli esperti del dominio che quindi possono validare il diagramma stesso fornendo una conferma al progettista sui requisiti individuati.

Fase 3: realizzazione (progettazione logica e fisica)

Prima avviene la progettazione **logica**: questa parte riguarda specificatamente la progettazione dei dati, e consiste nel determinare delle tabelle che verranno utilizzate per rappresentare i dati.

Poi avviene la progettazione **fisica**: realizzazione fisica delle tabelle attraverso appositi software e loro memorizzazione su un supporto di memoria.

Nelle applicazioni che abbiamo realizzato finora i dati venivano memorizzati nel file system del SO in forma testuale o binaria. Questo approccio è indicato con il termine approccio **file system**. Tale approccio è stato utilizzato fino agli ’80 del secolo scorso, è un approccio che è ancora possibile ritrovare in sistemi informatici datati (indicati con il termine **legacy**) o sistemi informatici in cui la quantità di dati da gestire è piccola.

Per grosse quantità di dati al giorno d’oggi si utilizza un approccio diverso, che si basa su sistemi software appositamente creati per la gestione dei dati. Tali sistemi software sono chiamati DBMS (Data Base Management System). I DBMS forniscono degli strumenti già pronti (che vedremo) per la gestione di dati. I DBMS sono nati negli anni ’70 del secolo scorso, inizialmente il costo di tali sistemi software era molto elevato e non alla portata di tutte le organizzazioni mentre attualmente la riduzione dei costi ha favorito la diffusione dell’approccio DBMS nella quasi totalità dei sistemi informatici di una certa complessità.

Quando l’anno scorso gli studenti hanno realizzato il progetto INFO-TPS, la memorizzazione dei dati è stata realizzata con un **approccio file-system**. In quel caso non era necessario utilizzare un DBMS perché i dati da memorizzare e le relative operazioni erano in quantità ridotte. In un’applicazione reale però, dove, ad esempio, fosse necessario memorizzare per un’attività commerciale i dati relativi ai clienti, ai fornitori, alle fatture, ai prodotti venduti, al magazzino ecc… e i diversi utenti dovessero avere bisogno si accedere ad informazioni diverse, estratte elaborando in maniera diversa tali dati, ecco che le funzionalità messe a disposizione da un DBMS risulterebbero estremamente utili. Non c’è un limite preciso sulla quantità di dati oltre la quale è opportuno introdurre il DBMS, comunque i DBMS sono strumenti molto potenti, si può dire che utilizzare un DBMS per una piccola applicazione sw sarebbe come utilizzare un bazooka per eliminare una mosca.

ASPETTI INTENSIONALE ED ESTENSIONALE DEI DATI

Questo capitolo serve per dare alcune importanti definizioni di concetti che saranno poi utilizzati in seguito.

Ricordiamo la differenza fra dato (descrizione di un fenomeno) ed informazione (incremento della conoscenza del fenomeno dato dall'elaborazione di dati).

Per elaborare i dati, e quindi estrarre informazioni, è necessario saperli interpretare, ossia sapere cosa significano.

Il valore assunto dai dati è chiamato aspetto estensionale.

Il significato dei dati è chiamato aspetto intensionale.

Esempio:

il seguente è un insieme di dati di cui si conosce solo l'aspetto estensionale, manca l'aspetto intensionale.

AB210CC		40		1000	
	80				
EA345JX		2000	CD512HL		
65				1200	

Cosa significano tali dati? Boh! Manca l'aspetto intensionale.

Per rappresentare i dati in modo che essi possano essere interpretabili, quindi per rappresentare l'aspetto intensionale ed estensionale si utilizzano generalmente delle **tabelle**.

Con la seguente tabella è possibile interpretare i dati poiché sono presenti entrambi gli aspetti.

Targa auto	Cilindrata	Potenza (kW)
CD512HL	1000	40
EA345JX	2000	80
AB210CC	1200	65

L'aspetto **intensionale** dei dati precedenti è fornito quindi dal seguente **schema (o intensione)**.

Targa auto	Cilindrata	Potenza (kW)

La tabella precedente (schema+valori) **rappresenta una possibile istanza dello schema**.

In informatica insiemi di molti dati che hanno la stessa interpretazione vengono generalmente rappresentati con delle tabelle e indicati con il nome di **categoria.** Per esempio la tabella precedente rappresenta la categoria delle automobili.

Generalmente i sistemi informatici sono caratterizzati da un numero ridotto di categorie con un numero molto elevato di dati, quindi il numero delle tabelle è trascurabile rispetto alla dimensione delle tabelle.

FILE DI DATI: APPROCCIO FILE SYSTEM

Come detto, in alcuni sistemi legacy o sistemi con pochi dati, è possibile che la memorizzazione dei record avvenga con l'approccio file system. Cerchiamo quindi di comprendere il suo funzionamento.

La prima cosa da chiarire è la differenza fra il concetto di file dal punto di vista **fisico** e dal punto di vista **logico**. Dal punto di vista **fisico** un file è una sequenza di byte quindi quando si parla di un file dal punto di vista fisico si intende "dove e come sono memorizzati i byte sul supporto fisco della memoria di massa", ad esempio in un SSD i byte sono memorizzati in blocchi distribuiti sulla superficie del disco anche in maniera non contigua.

Quando si parla di un file dal punto di vista **logico** si intende "come vengono interpretati i dati dall'applicazione che utilizza il file", ad esempio se il file ha memorizzato un insieme di automobili, dal punto di vista logico il file è una tabella in cui ogni riga (chiamata record) rappresenta un'automobile e ogni colonna (chiamata campo) un particolare aspetto intensionale del record memorizzato, come mostrato di seguito:

	Campo Targa auto	Campo Cilindrata	Campo Potenza(kW)
Record 1	CD512HL	1000	40
Record 2	EA345JX	2000	80
Record 3	AB210CC	1200	65

Cosa è un **file di dati**? E' un archivio di dati, ossia un insieme di dati **correlati** fra loro, identificato da un **nome**, **memorizzato** su un supporto di memoria di massa di un elaboratore che ha una **vita indipendente** dall'applicazione che l'ha creato (perché una volta creato esiste indipendentemente dal fatto che l'applicazione sia attiva o meno).

Operazioni sui file

Dal punto di vista logico le operazioni che si possono svolgere su un file sono le seguenti:

- **Inserimento**: aggiunta di nuovi record ai file
- **Modifica**: modifica dei valori dei record dei file
- **Cancellazione**: eliminazione di record dai file
- **Lettura**: lettura dei record dai file

Tali operazioni sono dette anche operazioni **CRUD** dalle iniziali dei termini inglesi che le identificano: **Create Read Update Delete**.

Organizzazione dei file

Per ogni file è possibile definire due tipi di organizzazione: un'organizzazione fisica e un'organizzazione logica:

- **organizzazione fisica**: indica il modo in cui sono memorizzati i file **fisicamente sulla memoria di massa**. Può essere sequenziale o diretta (detta anche casuale o random).
    - ***sequenziale*** indica la possibilità di accedere ad un record solamente "scorrendo" in sequenza tutti quelli che lo precedono.
    - ***diretta* (*casuale* o *random***) indica la possibilità di accedere ad un record specificandone la posizione (indirizzo).

  Questa caratteristica dipende solo dalla tecnologia con cui è realizzata la memoria di massa.
  Memorie di massa ad accesso sequenziale sono i **nastri magnetici** (simili alle musicassette o ai nastri vhs), attualmente poco utilizzate ma ancora presenti in sistemi **legacy** (ossia che utilizzano una tecnologia obsoleta).
  Memorie di massa ad accesso diretto sono tutte le memorie di massa attualmente più utilizzate: gli HDD, gli SSD. Questa modalità di accesso si dice random perché il tempo di accesso a qualsiasi indirizzo (quindi qualsiasi indirizzo scelto casualmente) è lo stesso, ciò equivale a dire che il tempo di accesso è indipendente dall'indirizzo in cui si trova l'informazione (anche se per l'HDD non è proprio vero al 100% a causa degli spostamenti dei dispositivi meccanici presenti in tale tecnologia).

- **organizzazione logica**: indica il modo in cui **un'applicazione** può accedere ai record del file. Può essere di tre tipi: **sequenziale**, **ad accesso diretto** (o random o casuale) o **indicizzata**.
  Se l'organizzazione logica è sequenziale (come, ad esempio, nei file di testo visti l'anno scorso), l'applicazione potrà "raggiungere" un record solamente partendo dal primo e "leggendo" tutti i record che precedono quello cercato. Quindi per accedere al record in posizione 6 bisogna prima che l'applicazione legga i 5 record precedenti.
  Se l'organizzazione è ad accesso diretto, l'applicazione potrà direttamente accedere al record in una determinata posizione (ad esempio il record in posizione 6), quindi con un solo accesso in memoria, rendendo l'operazione di lettura molto più veloce.
  L'organizzazione ad accesso indicizzato verrà spiegata successivamente poiché è un'evoluzione dell'accesso diretto.
  L'organizzazione logica di un file è determinata da come memorizza i dati sul file un'applicazione, tenendo però presente la seguente importante precisazione:
  se i file sono memorizzati su un supporto fisico **sequenziale**, l'unica possibilità di organizzazione logica è **sequenziale**, mentre se i file sono memorizzati su supporti ad accesso **diretto**, l'organizzazione fisica può essere **diretta oppure sequenziale**.

Parliamo un po' dell'**organizzazione logica** dei file di dati.

Perché è importante?

Per accedere a un record di qualsiasi file (per modificarlo, elaborarne i dati, eliminarlo…) è necessario ricercarlo. L'organizzazione logica influenza la modalità di ricerca (e quindi di accesso), consentendo o non consentendo algoritmi di ricerca più rapidi. La rapidità dell'accesso è un fattore fondamentale nelle prestazioni di un sistema informatico che contiene una grande quantità di dati. Si ricorda che le operazioni di accesso ai file (sia in lettura che in scrittura) sono **molto più lente** rispetto alle operazioni svolte nella memoria centrale (che, inoltre, è sempre ad accesso diretto). Vediamo come può avvenire l'accesso ad un record nei tre casi di organizzazione logica (sequenziale, diretta, indicizzata)

1. **organizzazione logica ad acceso sequenziale:**

   In questa organizzazione logica i record sono memorizzati uno di seguito all'altro e l'applicazione che legge i dati può accedere ai record solo in maniera sequenziale. Vediamo due possibili algoritmi di ricerca per questa tipologia di organizzazione logica.

   ESEMPIO:

   Dato un file contenente i seguenti record:

Nominativo	Telefono	Indirizzo	……
Rossi Mario	3390443456	Via del mare 12	…
Bianchi Giovanni	9893445694	Via Roma 14	…
Verdi Carla	5543443234	Piazza Garibaldi 21	…
Rossini Giuseppe	5443445678	Viale Marconi 1	…
Neri Maria	4567898765	Corso Mazzini 56	..
Bianchi Andrea	3425432313	Viale Italia 144	…
Neri Daniele	3421221232	Via Leopardi 5	..
……	…..	…..	…
…..	…..	…..	..
Rossi Maria	7834335629	Scali Manzoni 68	..

   Pseudocodice di un metodo per la ricerca di un record in file non ordinato

```
do
{
        recordLetto=leggiRecord();
        if (recordLetto. getNominativo==nominativoCercato)
                return recordLetto;
}
while (!file.EOF)
return null
```

   Accesso al file. Lento

L’esecuzione si interrompe quando viene individuato il record oppure alla fine del file

Se invece il file fosse ordinato in ordine alfabetico:

Nominativo	**Telefono**	**Indirizzo**	……
Bianchi Andrea	3425432313	Viale Italia 144	
Bianchi Giovanni	9893445694	Via Roma 14	
Neri Daniele	3421221232	Via Leopardi 5	
Neri Maria	4567898765	Corso Mazzini 56	
Rossi Maria	7834335629	Scali Manzoni 68	
Rossi Mario	3390443456	Via del mare 12	
Rossini Giuseppe	5443445678	Viale Marconi 1	
Vallar Paolo	5543443234	Piazza Posta 21	
Verdi Carla	5543443234	Piazza Garibaldi 21	
……	…	……	

L’algoritmo potrebbe essere migliorato interrompendo il ciclo nel momento in cui il nominativo record letto fosse maggiore del nominativo del record cercato:

```
do
{
        recordLetto=leggiFile();
        if (recordLetto.nominativo==nominativoCercato)
                return recordLetto.
}
while (!file.EOF && recordLetto.nominativo<nominativoCercato)
return null
```

L’esecuzione si interrompe quando viene individuato il record oppure quando il record letto è maggiore del record cercato. L’algoritmo di ricerca è più efficiente rispetto al caso di file non ordinato.

Quindi le caratteristiche dell’organizzazione logica sequenziale si possono così riassumere:

- e’ possibile accedere ai record solo in maniera sequenziale (abbiamo visto con i file di testo in Java)
- e’ l’unica possibile quando i file hanno un’organizzazione fisica sequenziale
- Se i record sono ordinati in base al campo di ricerca (ad esempio ordine alfabetico del nominativo) la ricerca può essere resa più efficiente perché può essere interrotta nel momento in cui si individua un record che ha un valore del campo di ricerca “maggiore” rispetto al record cercato. L’ordinamento va però effettuato sui record prima della memorizzazione sul file, quindi **la maggiore efficienza nella ricerca determina una**

maggiore complessità nell'inserimento dei record perché, in seguito ogni nuovo inserimento, è necessario un ordinamento.

2. **organizzazione logica ad acceso diretto (casuale, random):**

In questa organizzazione logica i record sono memorizzati uno di seguito all'altro ma ogni record è caratterizzato da un numero che ne specifica la posizione all'interno del file, come se fosse un array. Nell'applicazione che legge i dati è possibile accedere ad uno specifico record del file indicandone la posizione (In Java questo può essere fatto utilizzando la classe ***RandomAccessFile***, che non abbiamo visto). Quando si aggiunge un nuovo record, la sua posizione si ottiene incrementando di 1 la posizione dell'ultimo record memorizzato.
Caratteristiche di questa organizzazione logica sono le seguenti:

- E' possibile solamente con file ad organizzazione **fisica diretta.**
- E' possibile **solamente se i record hanno tutti la stessa lunghezza.**
- E' possibile anche per questi file accedere sequenzialmente (l'accesso sequenziale non è altro che un accesso diretto incrementando di 1 la posizione a cui si vuole accedere).

ESEMPIO 1: evidenziamo che è possibile accedere ad un record in una determinata posizione

	Nominativo	**Telefono**	**Indirizzo**	……
1	Rossi Mario	3390443456	Via del mare 12	…
2	Bianchi Giovanni	9893445694	Via Roma 14	…
3	Verdi Carla	5543443234	Piazza Garibaldi 21	…
4	Rossini Giuseppe	5443445678	Viale Marconi 1	…
5	Neri Maria	4567898765	Corso Mazzini 56	..
6	Bianchi Andrea	3425432313	Viale Italia 144	…
7	Neri Daniele	3421221232	Via Leopardi 5	..
……				…
……				..
n	Rossi Maria	7834335629	Scali Manzoni 68	..

Pseudocodice accesso diretto che consente di leggere il record in posizione 6 (Bianchi Andrea) come se il file fosse un'array:

recordLetto =leggiFile(6)

La possibilità di accedere direttamente ad un record in qualsiasi posizione però non porta grandi vantaggi nella ricerca di un record se i record non sono ordinati. Infatti generalmente un record non viene cercato in base alla sua posizione ma in base uno qualsiasi dei valori in esso memorizzati. La ricerca di un record di cui non è nota la posizione, in un file di record non ordinati, può avvenire quindi solo sequenzialmente Nello stesso modo visto per i file ad organizzazione sequenziale .

Il vantaggio nell'utilizzo dei file ad organizzazione logica ad accesso diretto si ha quando i record sono ordinati Secondo un qualsiasi campo di ricerca. In questo caso infatti è possibile utilizzare un algoritmo di ricerca più efficiente rispetto alla ricerca sequenziale: la ricerca binaria (o dicotomica), che verrà mostrata nel seguente esempio.

ESEMPIO 2: Algoritmo di ricerca **binaria** (o **dicotomica**) in file con record ordinati

	Nominativo	**Telefono**	**Indirizzo**	……
1	Bianchi Andrea	3425432313	Viale Italia 144	
2	Bianchi Giovanni	9893445694	Via Roma 14	
3	Neri Daniele	3421221232	Via Leopardi 5	
4	Neri Maria	4567898765	Corso Mazzini 56	
5	Rossi Maria	7834335629	Scali Manzoni 68	
6	Rossi Mario	3390443456	Via del mare 12	
7	Rossini Giuseppe	5443445678	Viale Marconi 1	
8	Vallar Paolo	5543443234	Piazza Posta 21	
9	Verdi Carla	5543443234	Piazza Garibaldi 21	
n	……	…	……	

La ricerca binaria (o dicotomica) funziona così:

- la posizione del record da confrontare con il valore cercato è sempre quella centrale dell'intervallo (valore "centrale"=(estremo inferiore+estremo superiore)/2)
- se i due valori coincidono il record è stato trovato
- se il valore "centrale" è < del valore cercato: rifaccio la ricerca nel semi-intervallo "destro" (quindi la posizione centrale+1 diventa l'estremo inferiore del nuovo intervallo di ricerca)
- se il valore "centrale" è > del valore cercato: rifaccio la ricerca nel semi-intervallo "sinistro"
  (quindi la posizione centrale-1 diventa l'estremo superiore del nuovo intervallo di ricerca)
- La ricerca termina quando il valore cercato viene individuato oppure quando l'estremo inferiore diventa maggiore dell'estremo superiore.

Nell'esempio della tabella precedente con 9 elementi si provi a "cercare" il record "Neri Maria" (posizione 4).

Pseudocodice della ricerca binaria (dicotomica)

```
estremoInf=1
estremoSup=n
while(estemoInf<=estremoSup)
{
    posizioneDaLeggere=(estremoInf+estremoSup)/2
    recordLetto =leggiFile(posizioneDaLeggere)
    if (recordLetto.getNominativo==nominativoCercato)
        return recordLetto
    else if (recordLetto.nominativo>nominativoCercato)
        estremoSup=posizioneDaLeggere-1
    else
        estremoInf=posizioneDaLeggere+1
}
return null;          //record non trovato
```

la complessità computazione della ricerca binaria è log2(n) (logaritmica) mentre di quella sequenziale è n (lineare), quindi la ricerca binaria è molto più efficiente.

Quindi le caratteristiche dell'organizzazione logica ad accesso diretto si possono così riassumere:

3. organizzazione logica indicizzata

Questa organizzazione è un caso particolare e evoluto dall'organizzazione logica ad accesso diretto . dunque è possibile solo quando i file sono ad accesso diretto. In questa organizzazione viene individuato un campo il cui valore deve essere univoco per ciascun record, tale valore viene chiamato **chiave**. Ad esempio un campo chiave per un file che contiene i dati di persone potrebbe essere il codice fiscale. Oltre ad identificare la chiave è necessario che venga realizzato ed associato al file che contiene i record (**chiamato file primario)** un ulteriore file chiamato **file indice**. Sia il file primario che il file indice devono essere ad accesso diretto.

- Il file primario contiene i record ed è un è un file ad accesso diretto (quindi anche l'organizzazione logica indicizzata è possibile solo se l'organizzazione fisica è diretta)
- I record vengono inseriti nel file primario sequenzialmente e questo file non viene ordinato.
- Il file indice è un file che contiene una tabella chiamata **tabella delle chiavi**. La tabella delle chiavi contiene ogni record contenuto nel file primario, ma per ogni record solamente due campi
    - la chiave
    - un valore che indica la posizione del record nel file primario.

e viene mantenuto ordinato in base alla chiave ogni volta che viene aggiunto eliminato o modificato un record .

Esempio di file primario:

	CF	**Nominativo**	**Telefono**	**Indirizzo**	……
1	rssmra123	Rossi Mario	3390443456	Via del mare 12	…
2	bncgvn213	Bianchi Giovanni	9893445694	Via Roma 14	…
3	vrdcrl987	Verdi Carla	5543443234	Piazza Garibaldi 21	…
4	rssgpn192	Rossini Giuseppe	5443445678	Viale Marconi 1	…
5	nramra121	Neri Maria	4567898765	Corso Mazzini 56	..
6	bncndr221	Bianchi Andrea	3425432313	Viale Italia 144	…
7	nrednl332	Neri Daniele	3421221232	Via Leopardi 5	..
……					…
……					..
n	rssmra878	Rossi Maria	7834335629	Scali Manzoni 68	..

Esempio di tabella delle chiavi dove la **chiave è il codice fiscale**:

bncndr221	**6**
bncgvn213	**2**
nramra121	**7**
nrednl332	**5**
rssmra878	**n**
rssmra123	**1**
rssgpn192	**4**
…….	
…..	
vrdcrl987	**3**

Si osservi che nella tabella delle chiavi, le chiavi sono **ordinate**. Ogni volta che si aggiunge un record al file primario, la tabella delle chiavi viene aggiornata e mantenuta ordinata.

Come avviene la ricerca e quale è il vantaggio di avere due file?

La ricerca di un record avviene nel seguente modo:

a. La tabella delle chiavi viene caricata dal file indice nella **memoria centrale**
b. il codice dell'applicazione effettua la ricerca del record (il nominativo) nella tabella delle chiavi in base alla chiave utilizzando un algoritmo efficiente (ad esempio ricerca binaria) individuando la posizione del record cercato nel file primario. Ad esempio ricercando il record con nominativo Rossini Giuseppe, la ricerca fornisce la posizione 4.
c. L'applicazione accede in maniera diretta al record cercato nel file primario.

Quale è il vantaggio? Detto così sembra che il record venga cercato due volte!

Questo è vero, ma il vantaggio sta nel fatto che la ricerca nella tabella delle chiavi avviene nella MC (memoria centrale), e tale operazione è molto più veloce rispetto all'applicazione di un algoritmo di ricerca direttamente sui record del file primario, record che sono, quindi, nella MM (Memoria di Massa).

Ma allora non posso direttamente caricare il file primario nella MC? Beh…se si potesse sarebbe senz'altro meglio effettuare la ricerca del record in questo modo, ma purtroppo i file primari sono file generalmente con più campi e molti record, quindi bisognerebbe caricare molti MB o addirittura GB nella MC, e questo non è possibile per problemi di spazio e di tempo.

Le tabelle delle chiavi sono invece tabelle di record piccole pertanto è possibile caricare tali dati nella MC (magari suddividendoli a loro volta in blocchi) ed effettuare su questi la ricerca in maniera veloce.

Un altro vantaggio è la possibilità di definire diversi file indice, quindi diverse tabelle delle chiavi ciascuna con una chiave diversa, come nel seguente esempio.

Esempio di tabella delle chiavi dove la **chiave è il numero di telefono.**

Si noti sempre che i record sono ordinati

3390443456	1
3421221232	7
3425432313	6
4567898765	5
5443445678	4
5543443234	3
7834335629	n
……..	..
9893445694	2

Questa tabella consente la ricerca utilizzando come chiave il numero di telefono.

La possibilità di creare molte tabelle degli indici, consente di realizzare la **multidimensionalità** dell'archivio, ossia di "vedere" il file ordinato in modi diversi, secondo criteri diversi.

BASI DI DATI E SISTEMI DI GESTIONE DELLE BASI DI DATI (DBMS)

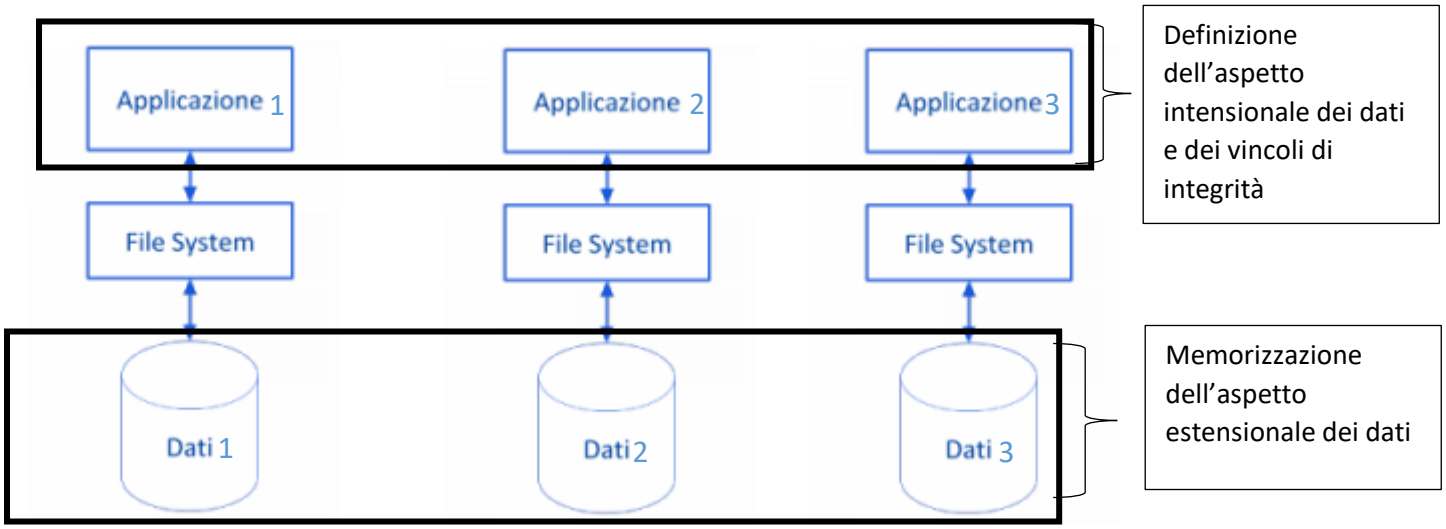
Premettiamo che per la gestione di una grande quantità di dati è ormai stato abbandonato l'approccio file system a favore di un approccio basato su **DBMS (Data Base Management System).** Le motivazioni di questo cambiamento e i vantaggi di questo approccio verranno ora illustrati.

Problematiche dovuto all'approccio file system

Nell'approccio file system, quello utilizzato da noi fino ad ora nella realizzazione delle applicazioni, nel codice di ogni applicazione realizzata venivano definite la struttura dei dati da utilizzare (ad esempio i campi dei record fattura: numero progressivo, data di emissione, importo ecc.), i vincoli di integrità sui dati (ad esempio non devono esserci fatture con lo stesso numero…), e le elaborazioni da svolgere sui dati (esempio: calcolo del fatturato nel mese di aprile.). I dati generati durante l'utilizzo dell'applicazione venivano poi memorizzati su un file di testo o su un file binario (aspetto estensionale) e recuperati dall'applicazione al momento del bisogno.

Nell'approccio file system dunque l'aspetto intensionale dei dati (ossia la loro struttura, il significato dei campi) e i vincoli di integrità **vengono definiti con il codice all'interno delle singole applicazioni**, e per ogni applicazione l'aspetto estensionale dei dati (i valori) viene memorizzato su un apposito file (o su più file).

Questa situazione si può rappresentare con la seguente figura:



Qual è il problema con questo approccio?

Il problema è che questo approccio **limita fortemente la condivisione di dati fra applicazioni diverse**.

Si pensi ad esempio ad un supermercato dove siano presenti un'applicazione "magazzino" per la gestione del magazzino, un'applicazione "casse" per la gestione delle casse, un'applicazione "contabilità" per la gestione degli ordini effettuati verso i fornitori.

Problema 1

L'applicazione "magazzino" ha bisogno di sapere la quantità di prodotto presente in magazzino e la quantità dello stesso prodotto presente sugli scaffali, in modo che quando la quantità presente sugli scaffali scende sotto un certo valore, l'applicazione notifichi all'utente (magazziniere) la necessità di riempire nuovamente gli scaffali.

Per l'applicazione "magazzino" i dati devono essere memorizzati nel seguente modo

Prodotto	Q,tità magazzino	Q.tità scaffale
Fagioli	1290	45
Biscotti	445	23
Pelati	770	32
..	….	…

L'applicazione "cassa", invece, serve All'utente "cassiere" per leggere, attraverso un lettore del codice a barre, quali e quanti prodotti sono stati acquistati da un cliente al fine di produrre il conto finale della spesa. Tale applicazione memorizzerà quindi i dati nel seguente modo:

Conto	Prodotto	Prezzo unitario	Q.tità venduta
1	Fagioli	1	3
1	Biscotti	2	1
1	Pelati	0,8	3
	..	….	…

Sarebbe molto utile che ogni volta che viene venduto un certo prodotto, la quantità venduta venisse sottratta alla "quantità scaffale". Per fare questo è necessario che l'applicazione "cassa" aggiorni anche il file dei dati dell'applicazione "magazzino". Anche l'applicazione "magazzino" deve poter modificare la "quantità scaffale". Cosa succede se entrambe le applicazioni intervengono contemporaneamente per aggiornare la quantità? Potrebbero accadere problemi di accesso concorrente che possono causare la perdita degli aggiornamenti.

Supponiamo inoltre che l'applicazione "contabilità" memorizzi i seguenti dati. Anche essa ha bisogno di conoscere la quantità di prodotti nel magazzino ed aggiorna tale quantità dopo ogni ordine

Prodotto	Q,tità acquistata	Fornitore	Q.tità Magazzimo
Fagioli	1000	Fagiolone spa	2290
Fagioli	500	Borlottone spa	2790
Pelati	2000	Capellone spa	2770
..	….	…	

Supponiamo che l'applicazione "contabilità" sia già presente nel supermercato e la proprietà del supermercato decide di far realizzare l'applicazione "magazzino" che attualmente non esiste. Il fatto

che le due applicazioni debbano entrambe aggiornare lo stesso dato rende molto complesso il lavoro del programmatore dell'applicazione "magazzino" perché è il programmatore dell'applicazione che deve, in qualche modo, accertarsi che i dati "quantità magazzino" siano coerenti nei due file, quindi l'applicazione magazzino dovrà andare ad aggiornare il file creato dall' applicazione "contabilità" ogni volta che i prodotti vengono portati dal magazzino allo scaffale, e l'applicazione "Contabilità" dovrà aggiornare il file prodotto da "magazzino" ogni volta che effettua un nuovo ordine. Risulta evidente che la condivisione dei dati rende molto più complessa l'attività del programmatore delle applicazioni.

Il succo del problema risiede nel fatto che **ogni applicazione definisce al suo interno le strutture dati (livello intensionale) "nel modo più comodo per essa", ma tale definizione delle strutture dati limita l'accesso ai propri dati da parte di altre applicazioni.**

Problema 2

Supponiamo che il supermercato decida di avviare una politica di marketing che prevede di regalare una quantità omaggio di alcuni prodotti quando la quantità acquistata supera un certo valore. Ad esempio per i fagioli e i pelati, ogni 3 prodotti acquistati ne verrà regalato uno in omaggio.

Per poter gestire questa politica di marketing è necessario apportare una modifica all'applicazione "cassa", in particolare andrà modificata la struttura dati che consente di calcolare il conto finale dell'applicazione "cassa" che potrebbe diventare la seguente:

Conto	Prodotto	Prezzo unitario	Q.tità venduta	Q.tità omaggio
1	Fagioli	1	3	1
1	Biscotti	2	1	0
1	Pelati	0,8	3	1
	..	….	…	

Per modificare la struttura dati sarà necessario ovviamente modificare il codice dell'applicazione "cassa". Se l'applicazione "magazzino" utilizza questo stesso file di dati per leggere la quantità venduta e aggiornare i dati del file "quantità magazzino/scaffale", a causa della modifica nella struttura dati dell'applicazione "cassa", anche il codice dell'applicazione magazzino dovrà essere modificato (la quantità da sottrarre allo scaffale non è più quella del campo quantità venduta ma la somma di "Quantità venduta"+ "Quantità omaggio")

Quindi ogni modifica alla struttura dati può generare la necessità di modificare tutte le applicazioni che utilizzano tali dati.

La causa di entrambi i problemi risiede **nel legame stretto fra dati e applicazione dovuto al fatto che i dati sono definiti ognuno nella propria applicazione** e quindi sono definiti in modo tale da massimizzarne l'efficienza per quella singola applicazione, rendendo difficile lo scambio di dati fra applicazioni diverse. Tale situazione si definisce **gestione non integrata** dei dati.

Soluzione grazie all'approccio DBMS

La soluzione ai problemi mostrati è quella di:

- "separare" il livello intensionale dei dati memorizzati (ossia la definizione delle strutture dati) dalla singola applicazione.
- raccogliere tutti i dati in un unico grande contenitore, detto **database** o **base di dati**;
- progettare il database in modo che i dati possano essere condivisi in maniera semplice da applicazioni diverse e su piattaforme diverse. Ogni applicazione "pesca" dal database solo i dati che "utilizza".

Questa soluzione è detta **gestione integrata dei dati** ed è resa possibile da un'apposita tipologia di sistemi informatici chiamati **DBMS (Data Base Management System)**

Potremmo quindi definire un database nel seguente modo:

un database è una collezione di dati PROGETTATI in modo tale da poter essere UTILIZZATI in maniera OTTIMIZZATA da DIFFERENTI APPLICAZIONI e UTENTI DIVERSI.

Un database è sempre realizzato e gestito da un DBMS.

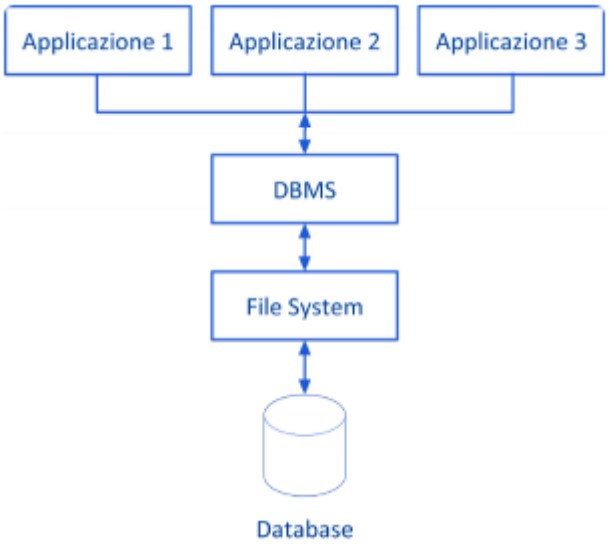
Definiamo quindi anche un DBMS:

Un DBMS (*DataBase Management System*) è un sistema software specializzato nell'archiviazione e nella gestione efficiente ed efficace di grandi moli di dati organizzati in *database* (o basi di dati).

Cosa consente di fare un DBMS? Consente ad una applicazione (o direttamente a un utente) la **creazione**, **manipolazione** e **interrogazione** dei dati garantendo tre cose:

- la **coerenza** dei dati (che i dati presenti nel database siano sempre quelli corretti),
- **il controllo degli accessi** (ossia che solo chi ha il permesso possa accedere ai dati).
- **il controllo della concorrenza** (ossia che l'accesso da parte di più utenti o applicazioni avvenga mantenendo la coerenza dei dati),

L'approccio DBMS della gestione dei dati si può rappresentare con la seguente figura, dove si evidenzia come il DBMS operi da interfaccia fra le piattaforme HW e SW su cui sono memorizzati i dati, e le applicazioni che utilizzano i dati.



E' il DBMS che si occupa di fornire alle applicazioni i dati di cui esse hanno bisogno. Il programmatore delle applicazioni è svincolato dalla necessità di progettare i dati, dovrà semplicemente "chiederli" al DBMS utilizzando un apposito linguaggio (SQL).

L'utilizzo di un DBMS garantisce **l'indipendenza logica** e **l'indipendenza fisica** dei dati.

Per **indipendenza logica** s'intende la possibilità di apportare modifiche alla struttura logica dei dati (ad esempio aggiungere campi) senza che le applicazioni debbano essere modificate. Per **indipendenza fisica** si intende la possibilità di modificare la struttura fisica del database (dove si trovano fisicamente i file, le modalità di accesso sequenziale o diretto ecc…) senza che la struttura logica (e di conseguenza le applicazioni) debba essere modificata.

Si precisa che il **concetto di integrazione** dei dati in un unico grande contenitore, che è implicito nella definizione di database, va tuttavia considerato solo da un *punto di vista logico*. Da un *punto di vista fisico*, infatti, un database potrebbe anche non essere composto da un unico file residente su un computer, ma al contrario potrebbe essere costituito da più file, allocati su più computer diversi e facenti parte di una rete i cui nodi si trovano fisicamente lontani. In questo caso si parla di **database distribuiti**.

Il linguaggio utilizzato per interagire con i database è un linguaggio standard chiamato **SQL** (Structured Query Language). Esso può essere utilizzato direttamente da un utente che vuole interagire con il database (ad esempio per "chiedere" o "aggiungere" dei dati) ma la maggior parte delle volte viene incorporato (si dice **embedded**, che significa incorporato) all'interno del linguaggio di programmazione con cui è realizzata l'applicazione (ad esempio Java o PHP).

I comandi di SQL si dividono in base al loro scopo in tre gruppi:

- DDL (Data Definition Language): linguaggio per definire la struttura dati del DB (creare le tabelle, i campi ecc..)
- DML (Data Manipulation Language): linguaggio per modificare i dati (ad esempio aggiungere record, eliminare record).

- QL (Query Language): linguaggio per l'interrogazione della base di dati, ossia per ottenere i dati dal database.

Quali sono gli utenti di un DBMS, ossia "chi" lo usa e per fare cosa?

- **Programmatori di applicazioni**: il programmatore, quando scrive il codice di un'applicazione integra dei comandi SQL nel proprio codice per **leggere o scrivere dei dati** (quindi il codice agisce solo sull'aspetto estensionale) nel database. Si dice che SQL è "**embedded**" ossia "incorporato" in un linguaggio di programmazione che può essere Java, PHP, C, ecc.
- **Utenti finali**: utilizzano il DBMS in maniera trasparente, senza saperlo, attraverso i software applicativi, per esempio un utente del registro elettronico.
- **Utenti amministratori** (DBA, DataBaseAdministrator): utenti professionali che definiscono e modificano la struttura del database (quindi agiscono sull'aspetto estensionale, ad esempio aggiungendo nuovi campi), e definiscono o modificano i diritti di accesso per gli altri utenti del DBMS.

ARCHITETTURA LOGICA DI UN DBMS (ARCHITETTURA ANSI-SPARC)

Quando una nuova tipologia di prodotto entra a far parte del mondo industriale e/o commerciale, nasce sempre l'esigenza di stabilire delle regole tecniche da far rispettare ai prodotti realizzati da diversi enti (società, laboratori di ricerca ecc..). Il rispetto degli standard da parte di un prodotto è una garanzia che tale prodotto si possa interfacciare con altri prodotti industriali presenti sul mercato. Ad esempio lo standard delle prese elettriche stabilisce dimensione e distanza fra i fori in modo da consentire l'interfacciamento con le spine di alimentazione dei dispositivi elettrici/elettronici.

Le caratteristiche di uno standard vengono generalmente stabilite da uno o più enti formati da esperti nell'ambito del prodotto da standardizzare.

Anche nell'ambito dei DBMS, quando negli anni '70 del secolo scorso iniziò la loro diffusione, venne avvertita nel mondo informatico questa esigenza di standardizzazione. Su proposta di un comitato americano chiamato SPARC (System Planning and Requirements Committee) nel 1972 il comitato X3 dell'ANSI (American National Standard Institute) ha iniziato a lavorare per la realizzazione di uno standard relativo ai DBMS, che è stato pubblicato nel 1978. Lo standard non impone delle caratteristiche fisiche o tecniche ma stabilisce l'architettura logica di un sistema informatico basato su DBMS che tutt'ora utilizzata (architettura ANSI/SPARC)

Il modello dell'architettura ANSI/SPARC stabilisce che un sistema di gestione basato su DBMS sia realizzato su tre livelli, che ora andiamo a spiegare.

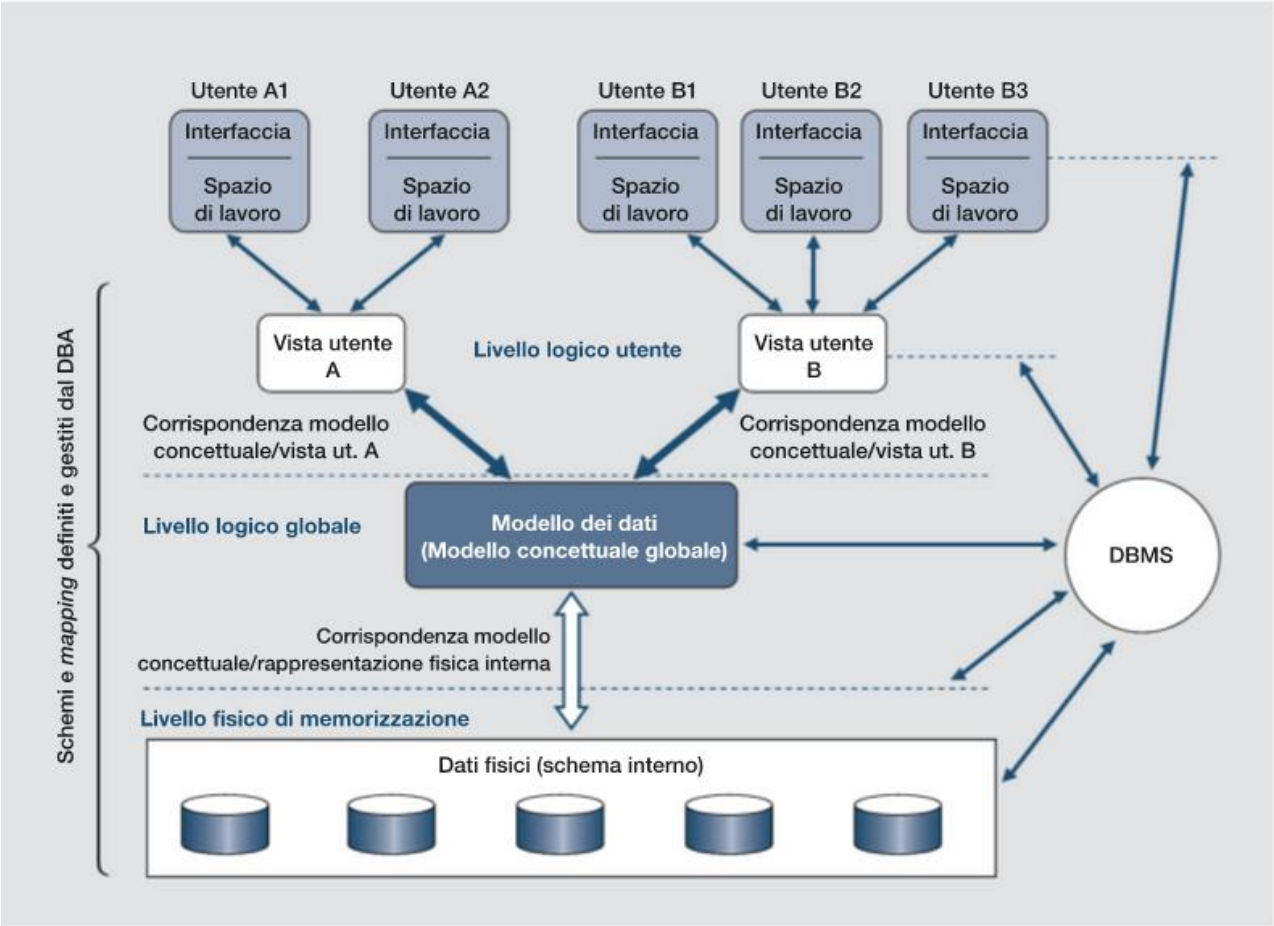
- **livello logico utente**: poiché ogni utente, utilizzando un' applicazione, ha la necessità di interagire NON con tutti i dati del DB ma solo un sottoinsieme di essi (come detto "non tutti gli utenti devono conoscere tutti i dati"), questo livello logico mette a disposizione di ciascuna applicazione solamente i dati di cui essa ha bisogno. Questo livello logico è realizzato introducendo il concetto di "vista". In prima approssimazione si può dire che una "Vista Utente" è un sottoinsieme del modello logico che mostra solamente i dati di interesse per un' applicazione o un utente.

Ad esempio per un supermercato l'applicazione "cassa" "vedrà" solamente i propri dati di interesse (prodotto, prezzo, quantità acquistata ecc.), l'applicazione "magazzino" "vedrà" solo i propri dati di interesse (Quantità in magazzino, quantità sullo scaffale ecc..).
Tali viste sono definite dall'amministratore del database (colui che gestisce il database) in base alle necessità degli utenti. (noi ci occuperemo di questo).

- **livello logico globale**: **definisce** la struttura logica dell'intero database, ossia l'aspetto intensionale e i vincoli di integrità di tutti i dati del DB. Viene realizzata dall'amministratore utilizzando il linguaggio DDL (noi ci occuperemo di questo).
- **livello fisico**: descrive come sono organizzati fisicamente i dati nella memoria di massa, ossia l'indirizzo fisico di memorizzazione dei dati, le modalità di accesso (sequenziale, indicizzato..), le strategie e gli algoritmi per ottimizzare la ricerca dei dati. Questa parte è a completo carico del DBMS, noi non ci occuperemo di questo poichè questi servizi ci verranno forniti in **maniera trasparente dal DBMS**.

Anche le operazioni di collegamento fra i vari livelli sono svolte dal DBMS in maniera trasparente per l'amministratore di DB. L'amministratore di DB deciderà di realizzare la struttura logica della base di dati (livello logico globale) e deciderà quali viste realizzare (livello logico utente), dopodiché "comunicherà" nel linguaggio SQL (sia con i comandi DML che DDL) al DBMS quello che vuole fare e il DBMS lo farà.

Il modello dell'architettura ANSI/SPARC:



DOMANDE GUIDA:

Definizione di dato, informazione, differenza fra sistemi informatici e sistemi informativi, progettazione concettuale

Progettazione logica e fisica

Aspetto intensionale ed estensionale

Cosa è uno schema (intensione).

Cosa è una categoria.

Tipi di organizzazione fisica dei file.

Tipi di organizzazione logica dei file (algoritmi di ricerca)

Approccio file system e approccio DBMS (quali problemi consente di superare l'approccio DBMS?)

Definizione di DMBS e di base di dati.

Indipendenza logica e indipendenza fisica

Linguaggio SQL (DDL, DML, QL)

I diversi utilizzatori di un DBMS

I 3 livelli dell'architettura ANSI/SPARC