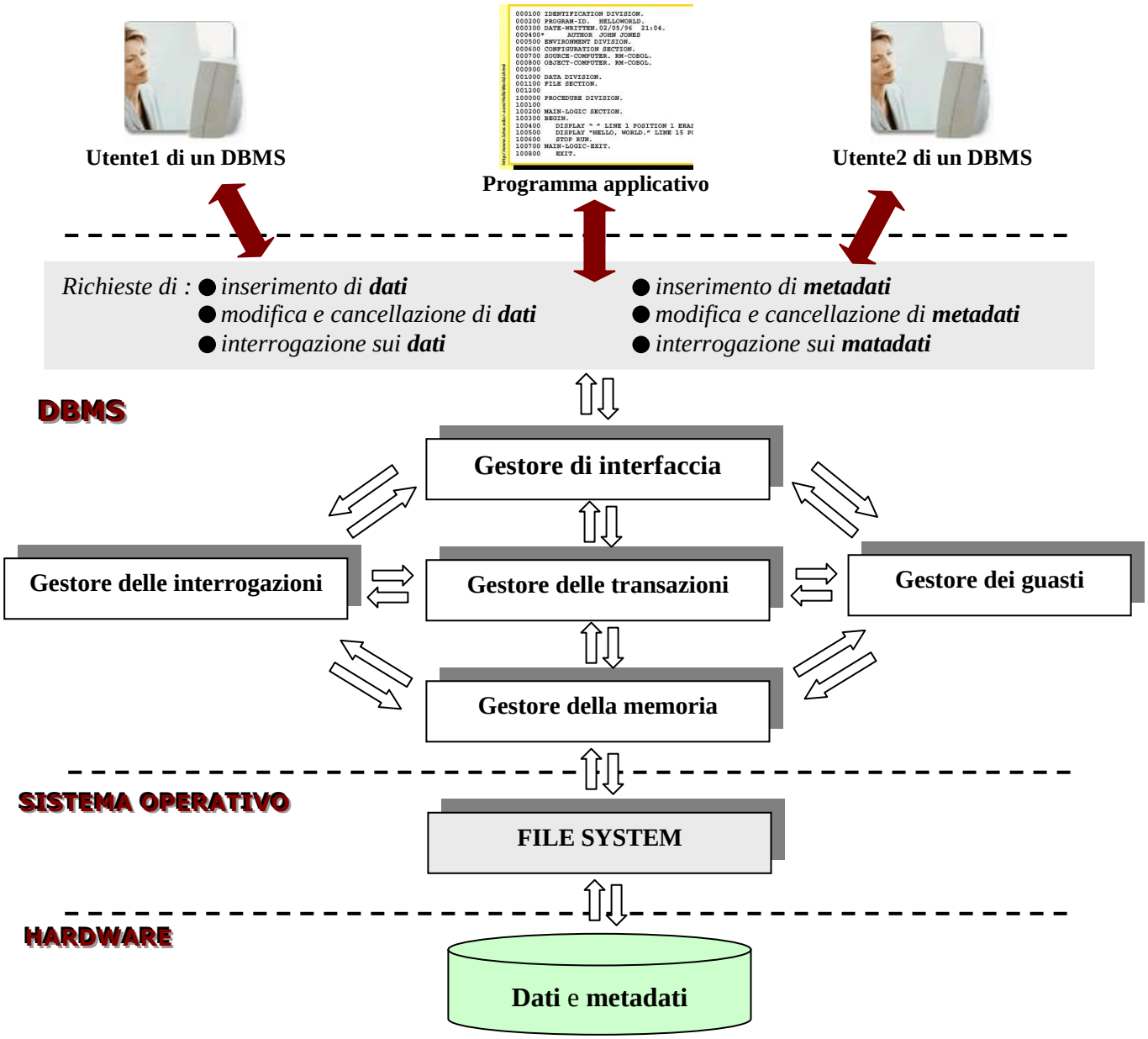
17. ARCHITETTURA DI UN DBMS

DEF: Il **DBMS (Data Base Management System)** è un **software** che, sulla base delle *specifiche utente*, offre la possibilità ad *utenti autorizzati* e nel rispetto di *regole prestabilite* di interagire con una base di dati ottenuta partendo da un progetto concettuale tradotto poi in un modello logico, e salvata su una memoria di massa.
Nel caso specifico di database relazionali si parla di **RDBMS** (**Relational Data Base Management System**).
Inoltre un **DBMS** si occupa anche della memorizzazione e della gestione, non solo di dati, ma anche di **metadati** ossia *di quelle informazioni relative alla struttura dei dati*

*Esempio: In una **relazione** ciascun singolo valore relativo ad un qualsiasi attributo è un **dato**, mentre invece il nome dell'attributo, la sua lunghezza, il suo tipo, ossia più in generale qualunque informazione relativa allo schema dei dati, è un **metadato**.*

Tutti i più importanti **DBMS** presenti sul mercato (come Oracle, MySql, PL/SQL, Microsoft Access, INFORMIX, etc.) hanno un'architettura interna che, a grandi linee, utilizza il seguente **schema funzionale**:



Ora analizzeremo in dettaglio i componenti individuati del **DBMS** dettagliati nel precedente schema:

● **Gestore dell'interfaccia:**

E' il componente di più **alto livello** poichè interagisce direttamente con l'utente della base dati, fornendo un'opportuna **interfaccia grafica** con la quale formulare richieste di operazioni sui **dati** o sui **metadati**.

Interagisce con il *gestore delle interrogazioni*.

Un **DBMS** può ricevere richieste di operazioni in una delle seguenti modalità:

- interattiva: direttamente attraverso opportuna interfacce utente o di tipo grafico (Graphic User Interface o GUI) oppure a linea di comando (Command Line Interface o CLI).
Queste interfacce permettono di eseguire le principali operazioni di inserimento, modifica, cancellazione e ricerca sia dei *dati* sia dei *metadati*.

*Esempio: molto utilizzata dai DBMS è l'applicazione web **phpMyAdmin** che consente in modalità GUI, attraverso un qualsiasi browser (Google Chrome, Internet Explorer, Safari, Mozilla Firefox etc. etc.), di **gestire un database MySQL** creando un database da zero, creando nuove tabelle ed effettuare operazioni che permettano di ottimizzarle, inserendo **nuovi utenti**, creando e modificando password e gestendo gli eventuali permessi che l'utente ha sul singolo database, etc. etc.*
*E' possibile **gestire un database MySQL** utilizzare anche una **consolle dei comandi** che, in modalità CLI, permette di eseguire tutte le operazioni sopra elencate svolte in modalità grafica da phpMyAdmin.*

Molte delle interfacce permettono di eseguire il cosiddetto **QBE** ossia **Q**uery **B**y **E**xample (anche lo stesso *phpMyAdmin*) grazie al quale l'utente può specificare i campi che intende visualizzare, nonché i criteri di interrogazione visualizzando l'output a video senza dover impostare nulla in SQL.

Esempio: In Microsoft Access c'è la composizione guidata dell'oggetto "query"

- batch: attraverso l'esecuzione di file di comandi scritti in SQL (istruzioni DDL ed istruzioni DML) o nei propri linguaggi (simil SQL).

*Esempio: con **Microsoft Access** possibile scrivere **macro** e **moduli**.*
*Le **macro** sono sequenze di comandi mentre i **moduli** sono procedure scritte in un linguaggio DDL e DML interno proprietario (VBA o Visual Basic for Application)*

- embedded: attraverso un apposito supporto per il linguaggio SQL (**client SQL** o **server SQL**) cui ci si può interfacciare, attraverso apposite librerie, utilizzando diversi linguaggi di programmazione sia lato server (PHP, JAVA con le Servlet, ASP, Python, etc.), sia lato client (JAVA con le Applet, JAVASCRIPT, etc.).

Il **client SQL** è un modulo software che permette di mandare in esecuzione operazioni sui dati scritte in SQL.

Il **server SQL** è invece un modulo software che viene utilizzato per il dialogo tra **DBMS** e **Web Server**.

- tramite programma applicativo: attraverso un vero e proprio ambiente di supporto per costruire programmi applicativi con interfaccia utente con i quali è possibile effettuare le operazioni sui dati

Esempio: Microsoft Access è ben integrato con l'ambiente di sviluppo VB ossia Visual Basic con il quale è possibile scrivere i programmi applicativi.

● Gestore delle interrogazioni **(o** Query processor**):**

Prende in input le query formalizzate a livello esterno dal *gestore dell'interfaccia* e le trasforma in una sequenza di richieste elementari da inviare al *gestore della memoria*.
La sequenza di interrogazioni elementari viene chiamata **piano di esecuzione delle interrogazioni o query plan.**

Le interrogazioni vengono espresse ad alto livello (ricordare il concetto di indipendenza dei dati) per ottenere insiemi di n-uple.
Una volta ottenuti i **dati** o i **metadati** , *il gestore delle interrogazioni* compone la tabella risultato finale e la restituisce al *gestore dell'interfaccia*.
Ovviamente tra i numerosi piani di esecuzione possibili per ogni query sottoposta il gestore delle interrogazioni sceglierà quello migliore ossia quello che riduce il numero di accessi alla memoria di massa dove sono immagazzinati i dati ed i metadati (ottimizzazione).

● **Gestore delle transazioni:**

Le principali funzioni del *gestore delle transazioni c*onsistono nel:

a) gestire le **operazioni consentite** attraverso opportune **autorizzazioni di accesso** alla base di dati:
b) gestire le **transazioni** ed il **ripristino** della base di dati;
c) gestire gli **accessi concorrenti** alla base di dati;

a) Le **operazioni consentite** possono essere classificate in:

(*) operazioni di *inserimento* di nuovi dati;
(*) operazioni di *modifica* dei dati inseriti;
(*) operazioni di *cancellazione* dei dati inseriti;
(*) operazioni di *lettura o interrogazione* dei dati inseriti;

(*) operazioni di *inserimento/modifica/cancellazione* dello schema dei dati o dei metadati.

Un DBMS utilizza una struttura dati detta **tabella di autorizzazione** nella quale sono registrati *gli utenti autorizzati*, le *operazioni consentite*, gli *oggetti* della base di dati sui quali possono operare.

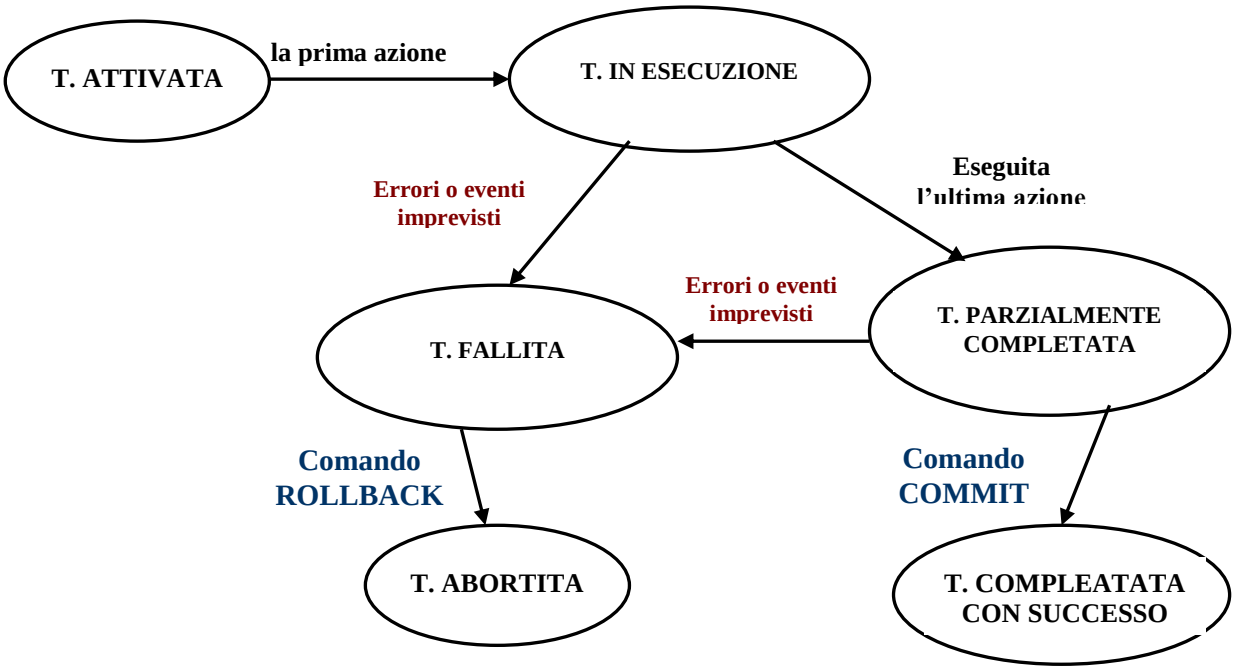
La gestione delle autorizzazioni ha come scopo quello di garantire la **sicurezza** dei dati legata a:

(*) **riservatezza** dei dati: per averla non tutti i dati devono essere accessibili a chiunque;
(*) **integrità** dei dati**:** I dati vanno difesi da modifiche non autorizzate, fraudolente e/o accidentali che possono renderli **corrotti** (ossia non leggibili), o che, violando qualche vincolo di integrità, possono renderli **inconsistenti** (dati duplicati con valori differenti) oppure **incongruenti** (dati le cui modifiche hanno tolto attinenza con la realtà di interesse descritta).

b)

DEF: Una **transazione** è una **operazione,** generalmente individuata da un *numero progressivo*, costituita da una successione di comandi/operazioni in esecuzione che forma un'unità logica di elaborazione sulla base di dati.
Affinchè una transazione *vada a buon fine* è indispensabile che tutte le operazioni elementari che la compongono siano eseguite con successo, altrimenti si considera *non eseguita*.
Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni è detto *sistema transazionale*

Gli **stati di avanzamento** (nel tempo) di una transazione possono essere riassunti grazie al seguente diagramma degli stati:



Dopo essere stata **attivata** *("started")*, una *transazione* si trova in *stato di esecuzione* nel quale vengono eseguite le azioni elementari di cui è composta.
Quando viene eseguita l'*ultima azione elementare* la *transazione* si dice *parzialmente completata*.
Se invece prima dell'esecuzione dell'ultima azione si verifica qualche errore o qualche evento imprevisto la *transazione* si dice **fallita** *("failed").*
Quando una *transazione* è parzialmente completata non è detto che diventi **completata con successo** *("committed")* poiché potrebbero intervenire imprevisti durante la fase di scrittura definitiva su disco.
Quando si verifica un *fallimento* vengono intraprese opportune azioni dette di **rotolamento all'indietro** *("rollback")* per arrivare allo *stato finale* nel quale la *transazione* si definisce **abortita** *("aborted")* annullando tutti gli effetti parziali prodotti dalle azioni elementari.

Prima di dichiarare il completo successo di una *transazione* occorre che il sistema abbia avuto il tempo di registrare una serie di informazioni **sul giornale delle modifiche** o **database log file** o più semplicemente **file di log**.

Nel file di log si registrano eventi riguardanti la *transazione* del tipo:
- è stata *attivata* la transazione <NomeTransazione>
- è stata *completata* la transazione <NomeTransazione>
- è *fallita* la transazione <NomeTransazione>

ma anche:
- è stata *inserita la riga* la <NuoviValori>
- è stata *cancellata la riga* la <ChiaveRegistrazione>>
- è stata *modificata la riga* la <VecchiValori> con <NuoviValori>

c) Una delle proprietà fondamentali di una transazione è **l'atomicità**.

Atomicità significa far apparire una transazione come un'unica azione **indivisibile**.
In questo modo le conseguenze delle azioni elementari che la compongono sono "visibili" sulla base dati solo nel caso in cui la transazione sia *completata con successo*.

SCENARIO: In generale in una base di dati accade che **più utenti** lavorino nello stesso momento **sugli stessi dati** attivando **più transazioni.**

E' dunque fondamentale che un DBMS gestisca la concorrenza degli accessi.

Gestire la concorrenza vuol dire assicurare che un accesso simultaneo da parte di più utenti agli stessi dati non comprometta la loro *integrità*, impedendo che gli stessi possano diventare *incongruenti o inconsistenti*.

PROBLEMA: Le due **situazioni anomale** più diffuse che potrebbero avere luogo in un sistema transazionale sono:

1) **situazione dell'ultima modifica persa** o **"last update":**

*Ipotizziamo due soli utenti e le relative transazioni chiamate **T1** e **T2** che agiscono sullo stesso dato **X** in modifica ed in modo concorrente.*
Può in teoria verificarsi la seguente situazione così riassunta.

1) **a1:** la *transazione* **T1 legge** il *dato* **X**
2) **b1:** la *transazione* **T2 legge** il *dato* **X**
3) **a2:** la *transazione* **T1 modifica** il *dato* **X**
4) **b2:** la *transazione* **T2 modifica** il *dato* **X**

La modifica al passo **3)** *verrà irrimediabilmente persa.*

2) **situazione della lettura sporca** o **"dirty read":**

*Ipotizziamo due soli utenti e le relative transazioni chiamate **T1** e **T2** che agiscono sullo stesso dato **X** in modifica ed in modo concorrente.*
Può in teoria verificarsi la seguente situazione così riassunta.

1) **a1:** la *transazione* **T1 legge** il *dato* **X**
2) **a2:** la *transazione* **T1 modifica** il *dato* **X**
3) **b1:** la *transazione* **T2 legge** il *dato* **X**
4) **a3:** la *transazione* **T1** abortisce e ripristina il vecchio valore del *dato* **X**
5) **b2:** la *transazione* **T2** modifica **X** sulla base del valore letto

*La modifica al passo **5)** si basa su di un valore del dato X che non esiste più in quanto sostituito dal precedente valore dalla transazione T1.*

SOLUZIONE: La più semplice e diffusa soluzione per il problema della concorrenza e la **tecnica di serializzazione** delle transazioni.

Questa tecnica, utilizzando il meccanismo di **blocco temporaneo o lock,** consente ad ogni singola transazione ***di avere la sensazione di operare in ambiente monoutente***.

Un **lock** è un *lucchetto* che il **DBMS** aggiunge ad ogni dato ed ad ogni elemento della base di dati, per evitare che altre transazioni possano utilizzarlo quando è già in uso da parte di una transazione.

Esistono **due tipi di lock**:

- exclusive Lock (***lock esclusivo***) o **lock di scrittura**: con questo tipo di lock nessuna transazione può ottenere un permesso di accesso al dato o elemento che interessa.

- shared lock (***lock condiviso***) o **lock di lettura**: con questo tipo di lock le altre transazioni possono accedere al dato o all'elemento ma considerato solo il lettura.

Per applicare la tecnica del lock è importante stabilire la **granularità** ovvero la *dimensione* degli oggetti e degli elementi della base di dati su cui il DBMS esercita il suo controllo.

Un **DBMS** può stabilire le seguenti tipologie di **granularità**:

- grossa: si aggiunge un lock al **file** contenente i dati (uno o più file corrispondono in genere ad una tabella);
- media: si aggiunge un lock al **record** del file contenente i dati;
- fine: si aggiunge un lock al **campo** del record del file contenente i dati;

Uno dei problemi più gravi che un DBMS deve affrontare nella gestione dei lock è quello relativo alle situazioni di **stallo** o di **attesa infinita** o **deadlock**.

Esempio: Consideriamo due transazioni T1 e T2 e due risorse R ed S.
Può in teoria verificarsi che

La *transazione* **T1** imposta un **lock** sulla *risorsa* **R**
La *transazione* **T2** imposta un **lock** sulla *risorsa* **S**

e contemporaneamente

La *transazione* **T1** imposta un **lock** sulla *risorsa* **S**
La *transazione* **T2** imposta un **lock** sulla *risorsa* **R**

In pratica entrambe le transazioni, dopo aver bloccato le rispettive risorse, richiedono per proseguire la risorsa dell'altra. In questo caso si è verificata una situazione di ***stallo o di attesa infinita*** *o* ***deadlock*** *perché ognuna attende l'altra.*

DOMANDA: Come è possibile risolvere una situazione di *stallo o di attesa infinita o deadlock*?

1) Una tecnica possibile (basata sul **LOCK MANAGER**) per risolvere il problema dello **stallo**, consiste nel far richiedere ad ogni transazione, prima dell'inizio della sua esecuzione, **tutti i lock necessari in una volta sola**.

In accordo a questa strategia, sarà un componente del DBMS, chiamato **lock manager,** a stabilire se concedere o meno alla transazione che ne fa richiesta, i lock su tutte le risorse necessarie alla sua esecuzione, consultando **una tabella dei lock** per vedere se esistono altre transazioni che hanno già ottenuto qualche lock su qualche risorsa presente nelle lista, che non sia stata ancora rilasciata.

Quindi riassumendo il lock manager, quando una transazione ne fa richiesta potrà:

- **concedere tutti i lock necessari**, se non esiste, nella tabella dei lock, alcuna risorsa già in stato di lock (quindi bloccata) perché richiesta da un'altra transazione in esecuzione;
- **negare tutti i lock necessari** perché una o più risorse necessarie alla sua esecuzione risultano già presenti nella tabella dei lock e quindi risultano bloccate da un'altra transazione in esecuzione.

2) Un'altra tecnica possibile (chiamata **ABORT AND RESTART**) per risolvere il problema dello **stallo**, consiste nel far abortire forzatamente una delle due transazioni.
La transazione abortita dovrà obbligatoriamente:
- ripristinare lo stato del sistema (nel nostro caso della base dati) prima della sua esecuzione
- essere ripresentata con una priorità molto alta per essere immediatamente servita dal sistema.

- **Gestore della memoria:**
- **Gestore dei guasti:**

PREMESSA

Prima di poter parlare del *gestore della memoria* e del *gestore dei guasti* dobbiamo accennare qualcosa circa l'ultima fase della progettazione di una base di dati ossia quella della **progettazione fisica**.

Come sappiamo la fase della **progettazione fisica** prende in *input* il *modello logico relazionale* prodotto durante la *fase della progettazione logica* e produce in *output* uno ***schema fisico*** della base di dati.

Quest'ultimo schema dipenderà dal DBMS che si vuole utilizzare e per questa ragione occorre scegliere quello più adatto al nostro progetto valutando al risposta ai seguenti fattori discriminanti:

- **efficienza in tempo**: occorre prevedere i *tempi di risposta* alle operazioni richieste che devono rientrare in determinati parametri anche tenendo conto del *carico applicativo* (ossia del traffico esterno al sistema);

- **efficienza in spazio**: occorre prevedere il *volume medio* dei dati ed il loro *tasso di crescita*;

- **ambiente di utilizzo**: occorre prevedere il tipo di interazione con il DBMS e la *piattaforma* su cui vogliamo operare.

Inoltre occorre valutare che alcuni DBMS si adattano meglio a gestioni "locali" altri meglio a gestioni "in rete"

Infine occorrerà decidere:

- i **dispositivi di memorizzazione** o "device" su cui memorizzare i dati (dischi, nastri);

- gli **indici** per accelerare i tempi di *ricerca* e di *ordinamento* dei dati;

- le **tecniche di organizzazione** degli archivi sulle memorie di massa;

- i **metodi di accesso** agli archivi;

- i **fattore di blocco** ossia il numero di record che vengono trasferiti in un'unica operazione dalla memoria di massa alla memoria centrale.

- **Gestore della memoria:**

E' il componente del DBMS più strettamente legato alla progettazione fisica di una base di dati *(ecco il perché della PREMESSA che abbiamo effettuato precedentemente)*

E' responsabile della definizione di alcuni parametri (sia qualitativi, sia quantitativi) relativi alle *strutture fisiche di memorizzazione dei dati*.

I dati che sono *logicamente* memorizzati in **tabelle** vengono *fisicamente* memorizzati su **file**.

In genere si tende a mettere in relazione:

Struttura logica di un dato		**Struttura fisica di un dato**
ATTRIBUTO	⇨	CAMPO
ENNUPLA	⇨	RECORD
RELAZIONE	⇨	FILE

mentre **ciò non è sempre vero** poiché per la necessità di ottimizzazione dello spazio occupato nella memoria di massa e per velocizzare al massimo gli accessi, **la strutturazione fisica dei dati** non trova corrispondenza esatta (ossia uno ad uno) con **la strutturazione logica dei dati.**

Ciò accade specialmente riguardo la relazione: per le ragioni sopra esposte, una relazione potrebbe essere memorizzata su più file

Il compito principale del **gestore della memoria** è recuperare o modificare i **blocchi dei dati presenti nei file** su disco. *(ricordiamo che un **blocco** è l'unità minima di trasferimento dei dati dalla memoria di massa alla memoria centrale e viceversa).*

Per svolgere questo compito il gestore della memoria si affida ai seguenti due suoi **sottocomponenti**:

- **gestore dei file su disco**: che si occupa di *recuperare* i blocchi di dati dai dischi su richiesta del *gestore del buffer di memoria*.

- **gestore del buffer di memoria**: che si occupa di *mappare* i blocchi di dati proveniente dal *gestore dei file* in **pagine** di memoria centrale;

Il responsabile delle scelte di organizzazione dei dati su memoria di massa è sempre il **DBA** (**D**ata **B**ase **A**dministrator) che utilizzerà il **DMCL** (**D**evice **M**edia **C**ontrol **L**anguage) o il **DSL** (**D**ata **S**torage **L**anguage) per le necessarie impostazioni e le opportune configurazioni dei dispositivi.

● Gestore dei guasti:

Ha il compito di *conservare nel tempo* la base di dati (***persistenza della base di dati***)

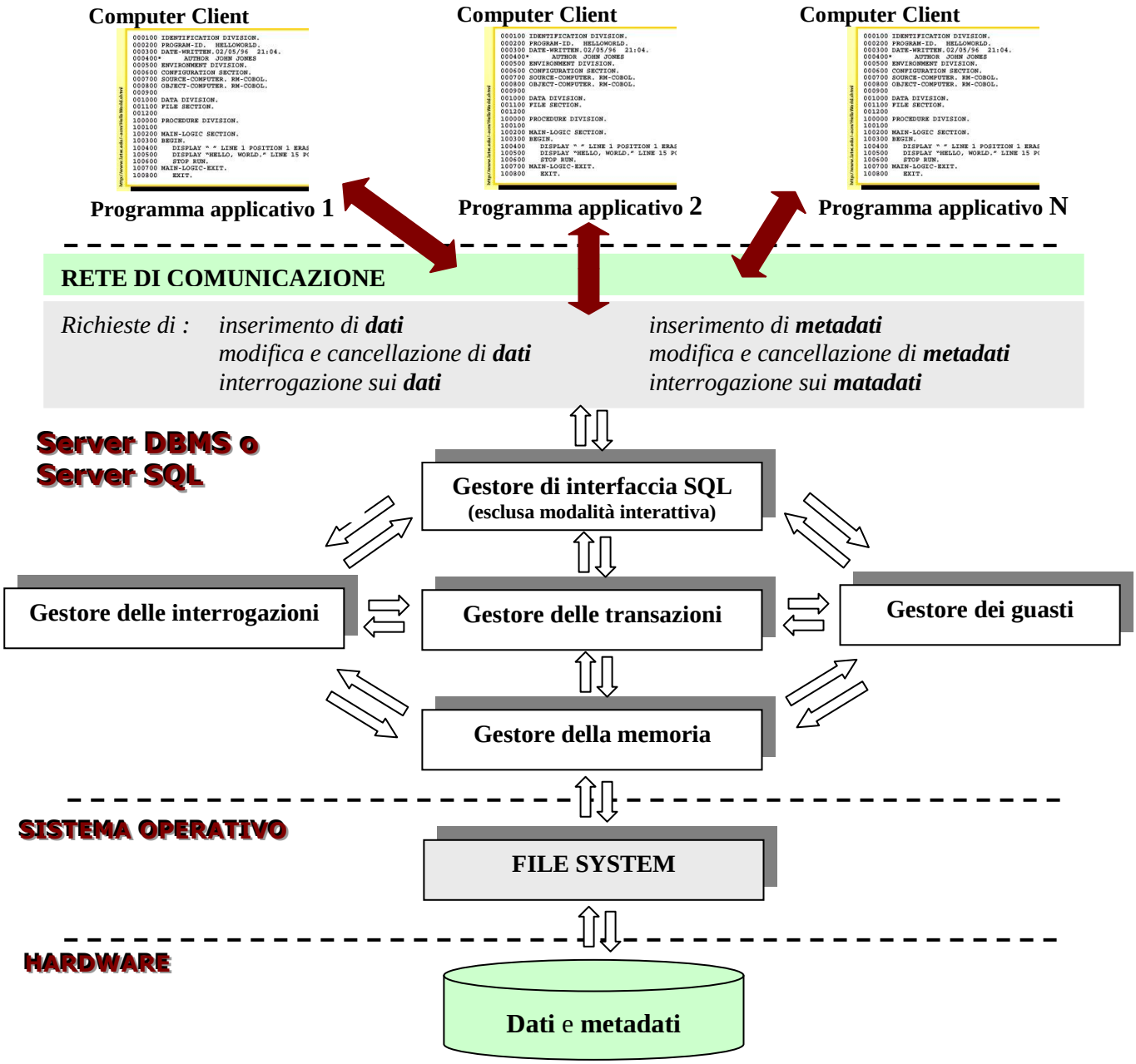
Fa fronte ai seguenti tipi di guasti:

- **guasti di sistema**: interruzioni del funzionamento dei dispositivi hardware per *cali di tensione* o *errori del software del sistema operativo* (i cosiddetti *bug*).
Possono causare l'inconsistenza dei dati producendo perdite di dati nei buffer tra memoria centrale e memoria di massa;

- **guasti di dispositivo**: guasti che riguardano i dispositivi di memoria non volatile ossia le memorie di massa *(ad esempio crash del disco rigido).*

Entrambi i tipi di guasti possono causare la perdita della base di dati.

ARCHITETTURA CLIENT-SERVER DI UN DBMS



L'**architettura client-server** è alla base di tutti i principali prodotti DBMS presenti sul mercato.

L'intero DBMS è in pratica un *server* in quanto riceve richieste di servizi (ossia richieste di operazioni sui dati) e risponde eseguendo tali operazioni restituendo tabelle come risultato.

In tale schema architetturale occorre escludere la parte del gestore di interfaccia che riceve interrogazioni poiché si suppone che l'interazione tra **server DBMS** (spesso anche chiamato **server SQL**) e le applicazioni *client* avvenga quasi esclusivamente tramite *la parte di supporto per SQL* della componente *gestore di interfaccia*.

DBMS ATTIVI

Una base di dati si dice **attiva** quando è possibile definire e gestire regole attive dette anche **trigger**.
Una **regola attiva o trigger** è un'entità che segue il paradigma **Evento-Condizione-Azione** ossia ciascuna regola reagisce a determinati *eventi* valutando una *condizione*.
Se essa è vera esegue *l'azione* associata.

L'esecuzione dei trigger viene gestita da una componente autonoma chiamata **processore delle regole o *rule engine*** che è una componente sempre in attesa che si verifichino eventi tali in modo da poter eseguire le relative regole sempre dopo aver verificato le condizioni associate.

In una base dati **attiva** possiamo avere quindi due tipi di transazioni:
- **transazioni eseguite dall'utente**;
- **transazioni eseguite dalle regole attive o trigger**.

Per questo motivo un *DBMS attivo* si dice che ha un comportamento **reattivo** differente da quello detto **passivo** di un *DBMS tradizionale*.