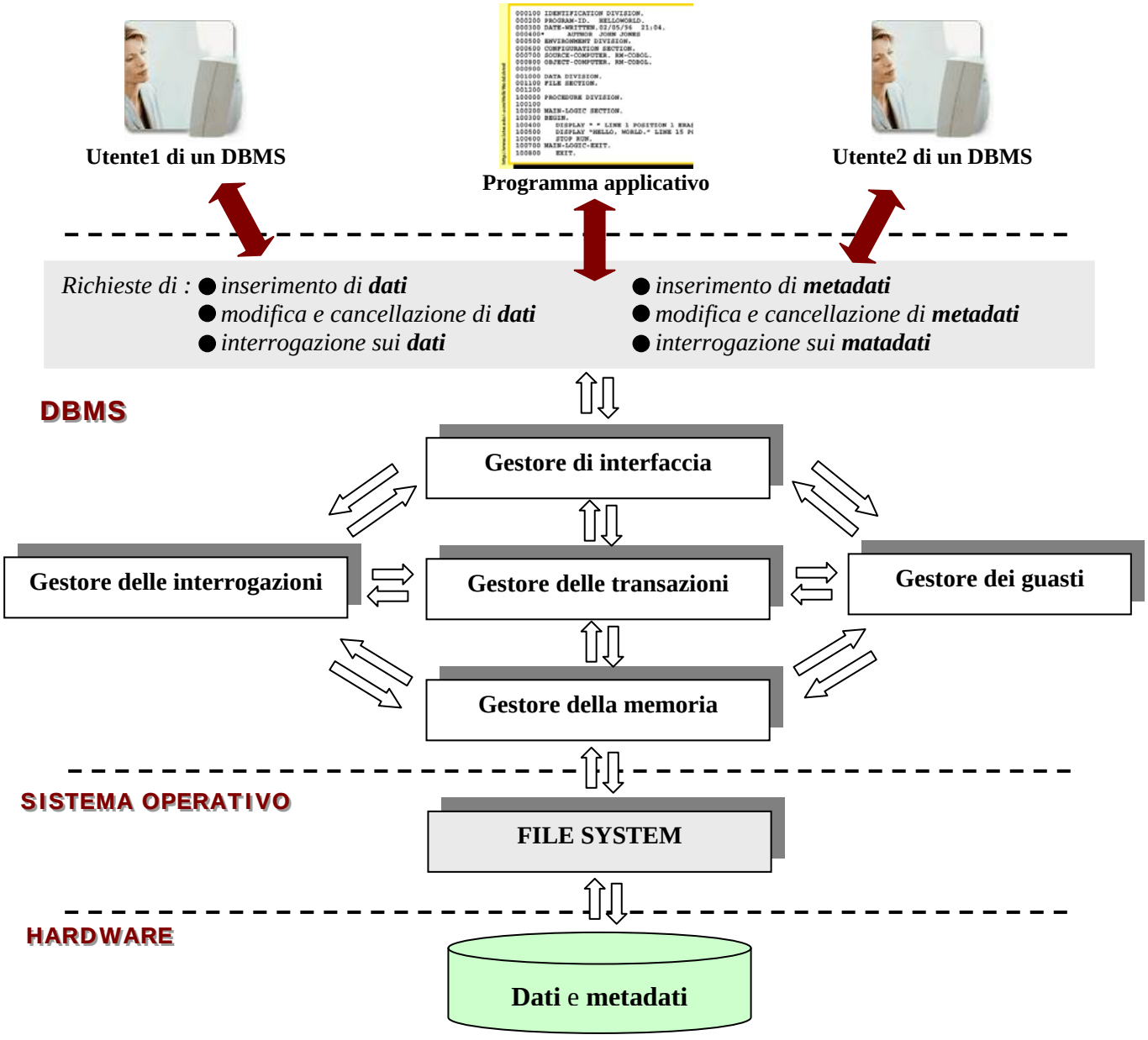
17. ARCHITETTURA DI UN DBMS

Vediamo ora dove sono effettivamente memorizzati i dati e quali sono le principali funzionalità di un sistema di gestione di basi dati.
Parliamo quindi di **DBMS**.

DEF: Un **DBMS** è un **insieme di programmi** che permettono di *interagire* con una base di dati (ottenuta partendo da un progetto concettuale tradotto poi in un modello logico) *consentendo operazioni* agli utenti autorizzati, nel rispetto di *regole* stabilite.

Un **DBMS** si occupa della memorizzazione e della gestione non solo di dati ma anche di **metadati** ossia *di informazioni sulla 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**.*

I più importanti **DBMS** presenti sul mercato (come Microsoft Access, Oracle, MySQL, etc.) hanno un'architettura interna che, a grandi linee, utilizza il seguente **schema**:



Ora analizzeremo in dettaglio i componenti individuati del DBMS:

● **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 grafiche. Queste interfacce permettono di eseguire le principali operazioni di inserimento, modifica, cancellazione e ricerca sia dei *dati* sia dei *metadati*.
Uno strumento molto utilizzato dai DBMS è il **QBE** ossia **Q**uery **B**y **E**xample 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 o nei propri linguaggi (DDL e DML).
*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 **V**isual **B**asic for Application)*

- 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 **V**isual **B**asic con il quale è possibile scrivere i programmi applicativi.*

- tramite Client SQL: attraverso un apposito supporto per il linguaggio SQL (client SQL o Server SQL).
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.

● **Gestore delle interrogazioni:**

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*.
Una volta ottenuti i **dati** o i **metadati** , *il gestore delle interrogazioni* compone la tabella risultato finale e la restituisce al *gestore dell'interfaccia*.
La sequenza di interrogazioni elementari viene chiamata **piano di esecuzione delle interrogazioni o query plan.**
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.

● **Gestore delle transazioni:**

Le principali funzioni del *gestore delle transazioni* consistono nel:
a) gestire le **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 *modifica* dello schema dei dati o *modifica* dei metadati.

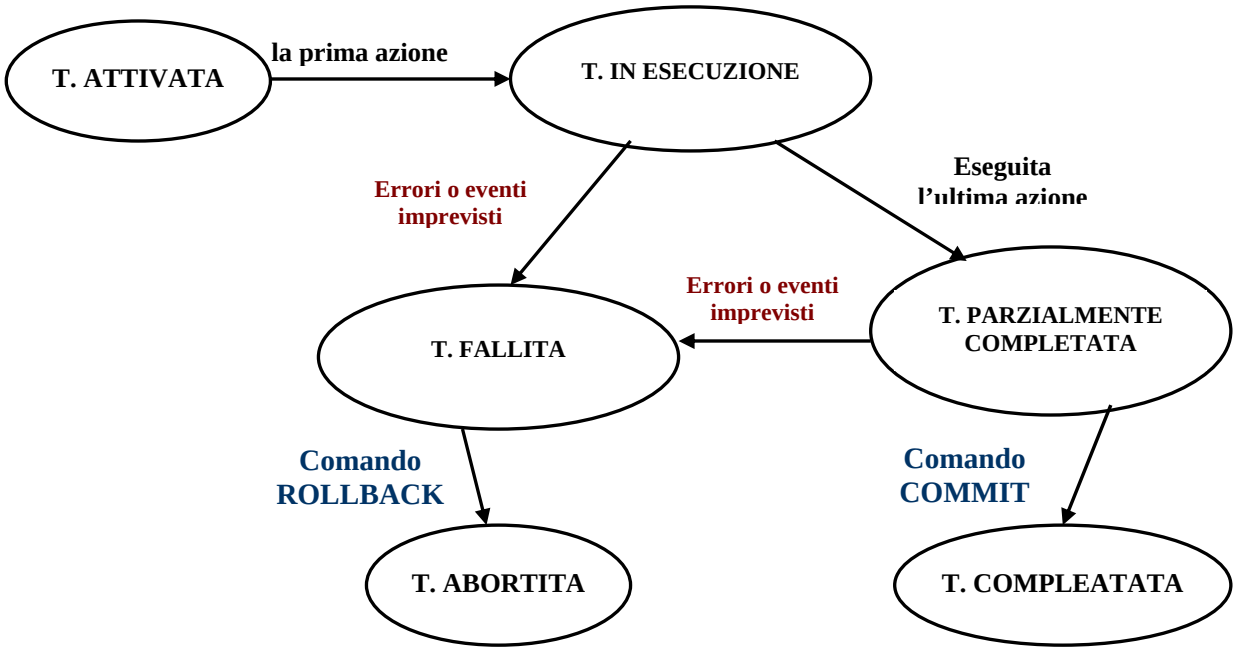
Un DBMS utilizza una struttura dati detta **tabella di autorizzazione** nella quale sono registrati *gli utenti autorizzati*, le *operazioni* che possono compiere, gli *oggetti* della base di dati sui quali possono operare.

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), oppure **non ammissibili o non corretti** (ossia che violano qualche vincolo di integrità), oppure **inconsistenti** (dati duplicati con valori differenti) oppure **non attinenti** (dati le cui modifiche hanno tolto attinenza con la realtà di interesse descritta).

b)

Una **transazione** è una **operazione,** generalmente individuata da un *numero progressivo,* costituita da una serie di operazioni elementari che devono essere necessariamente tutte eseguite per *andare a buon fine*, altrimenti si considera *non eseguita*.

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



Dopo essere sta **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 del'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")* o 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 solo in caso la transazione venga *completata con successo*.

In una base di dati in generale accade che **più utenti** lavorino nello stesso momento **sugli stessi dati** attivando **più transazioni.**
E' fondamentale che un DBMS gestisca la concorrenza degli accessi.

> **Gestire la concorrenza** vuol dire assicurare che un accesso simultaneo da parte di più utenti non comprometta l'*integrità dei dati,* impedendo che gli stessi possano diventare *corrotti o inconsistenti*.

Le due **situazioni anomale** più diffuse 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 verificarsi la seguente situazione così riassunta.

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

La modifica al passo) 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 verificarsi la seguente situazione così riassunta.

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

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

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 DBMSD può stabilire le seguenti granularità:
- grossa: si aggiunge un lock al **file** contenente i dati;
- 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**.

Esempio: Consideriamo due transazioni T1 e T2 e due risorse R ed S. Può succedere che

T1 lock **R**
T2 lock **S**

e contemporaneamente

T1 richiede lock su **S**
T2 richiede lock su **R**

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*** *perché ognuna attende l'altra.*

1) Una tecnica per risolvere il problema dello **stallo** consiste nel far richiedere ad ogni transazione tutti i lock in una volta sola, prima dell'inizio della sua esecuzione.
Così un componente del DBMS chiamato **lock manager** stabilisce se concedere o meno i lock su tali risorse consultando **una tabella dei lock** per vedere se esistono altre transazioni che hanno già ottenuto qualche lock su qualche risorsa richiesta e non l'hanno ancora rilasciata.

2) Un'altra tecnica (chiamata **abort and restart**) per risolvere il problema dello **stallo** consiste nel far abortire forzatamente una delle due transazioni. La transazione abortita dovrà ripristinare lo stato del sistema prima della sua esecuzione ed essere ripresentata con una priorità molto alta per essere immediatamente servita dal sistema.

LA PROGETTAZIONE FISICA DI UNA BASE DI DATI

Prima di 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**.

La fase della **progettazione fisica** prenderà in *input* lo *schema relazionale* prodotto durante la *fase della progettazione logica* e darà 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"

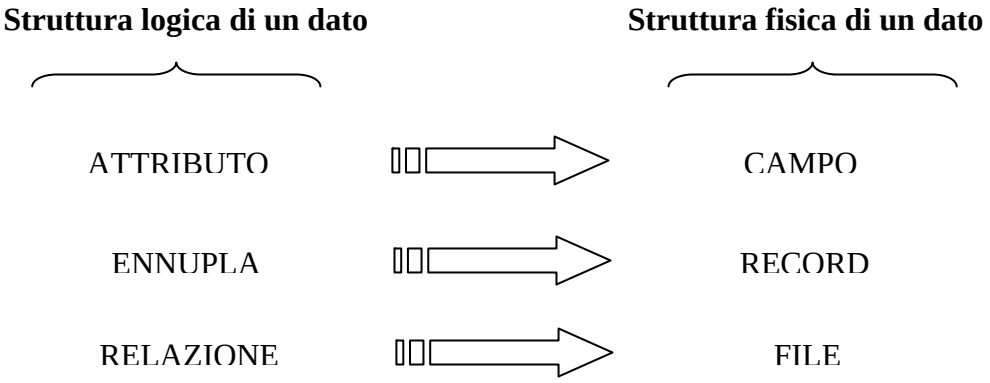
Infine occorre decidere:
- i **dispositivi di memorizzazione** o "device" su cui memorizzare i dati (dischi, nastri);
- **gli indici** per accelerare i tempi di *ricerca* e di *ordinamento* dei dati;
- le **tecniche di organizzazione** degli archivi sulle memorie di massa;
- i **metodi di accesso** agli 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.
E' responsabile della definizione di alcuni parametri (qualitativi e quantitativi) relativi a *strutture fisiche di memorizzazione dei dati*.

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

In genere si tende a mettere in relazione:



mentre **<u>ciò non è sempre vero</u>** 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.**
(Ad esempio per le ragioni sopra esposte, una relazione potrebbe essere memorizzata su più file)

Il compito 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 **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 su memoria di massa è sempre il **DBA** (**D**ata **B**ase **A**dministrator) che utilizza il **DMCL** (**D**evice **M**edia **C**ontrol **L**anguage) o il **DSL** (**D**ata **S**torage **L**anguage) per le impostazioni e le configurazioni dei dispositivi.

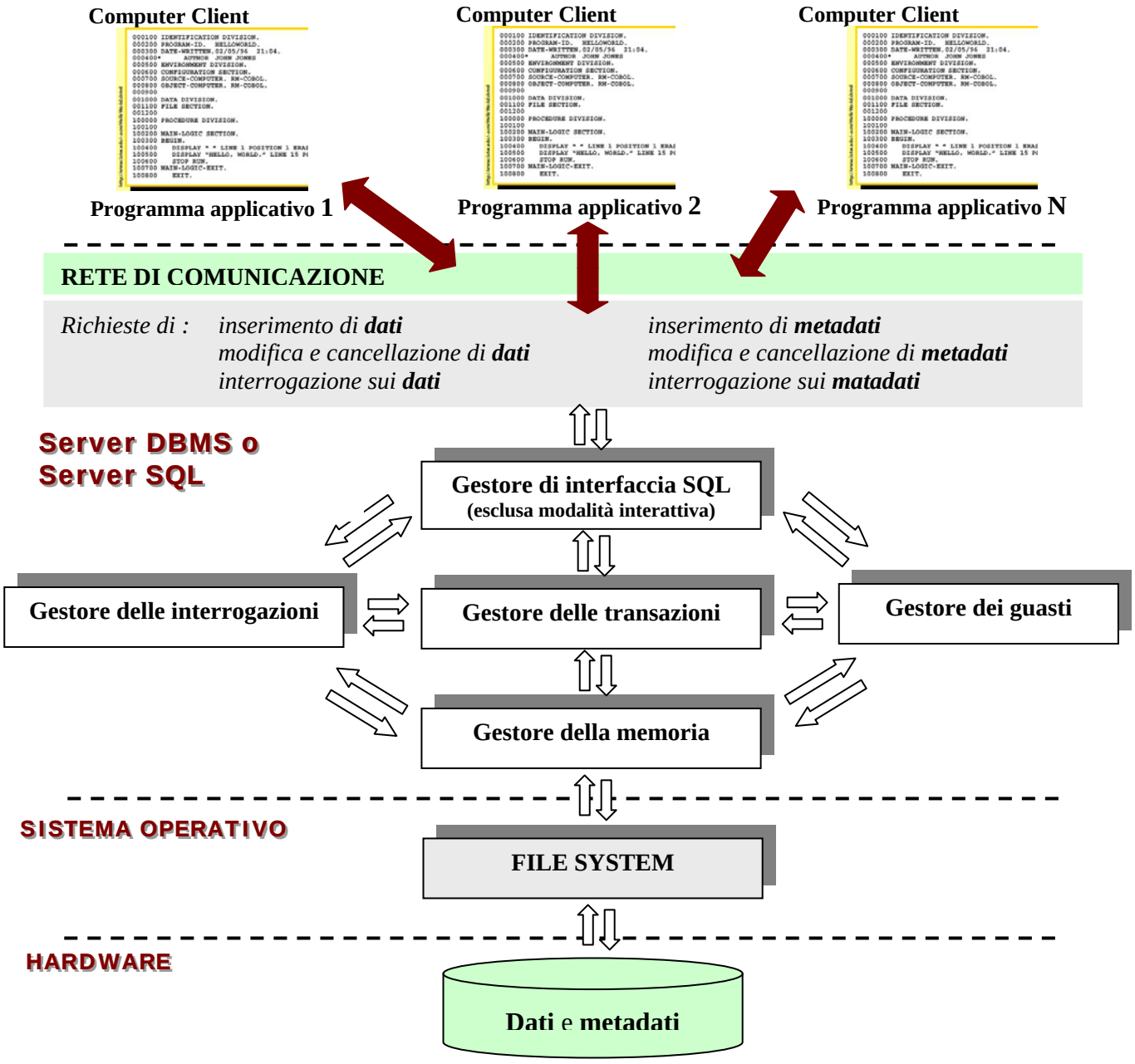
● **Gestore dei guasti:**

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 (*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 (memoria di massa) (ad esempio crash del disco rigido).
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*.