



## Funzionalità DBMS

Annalisa Franco, Dario Maio  
Università di Bologna

## II DBMS

- Un DataBase Management System (DBMS) è un insieme di programmi che permettono agli utenti di definire, costruire, manipolare e condividere una base di dati.

DEFINIRE	COSTRUIRE	MANIPOLARE	CONDIVIDERE
<ul style="list-style-type: none"><li>• Specificare i tipi di dato</li><li>• Descrivere la struttura dei dati</li><li>• Descrivere i vincoli</li></ul>	<ul style="list-style-type: none"><li>• Immagazzinare i dati in un supporto di memorizzazione gestito dal DBMS stesso</li></ul>	<ul style="list-style-type: none"><li>• Interrogare il database</li><li>• Aggiornare i dati</li></ul>	<ul style="list-style-type: none"><li>• Consentire a più utenti di accedere in modo concorrente alla stessa base di dati</li></ul>

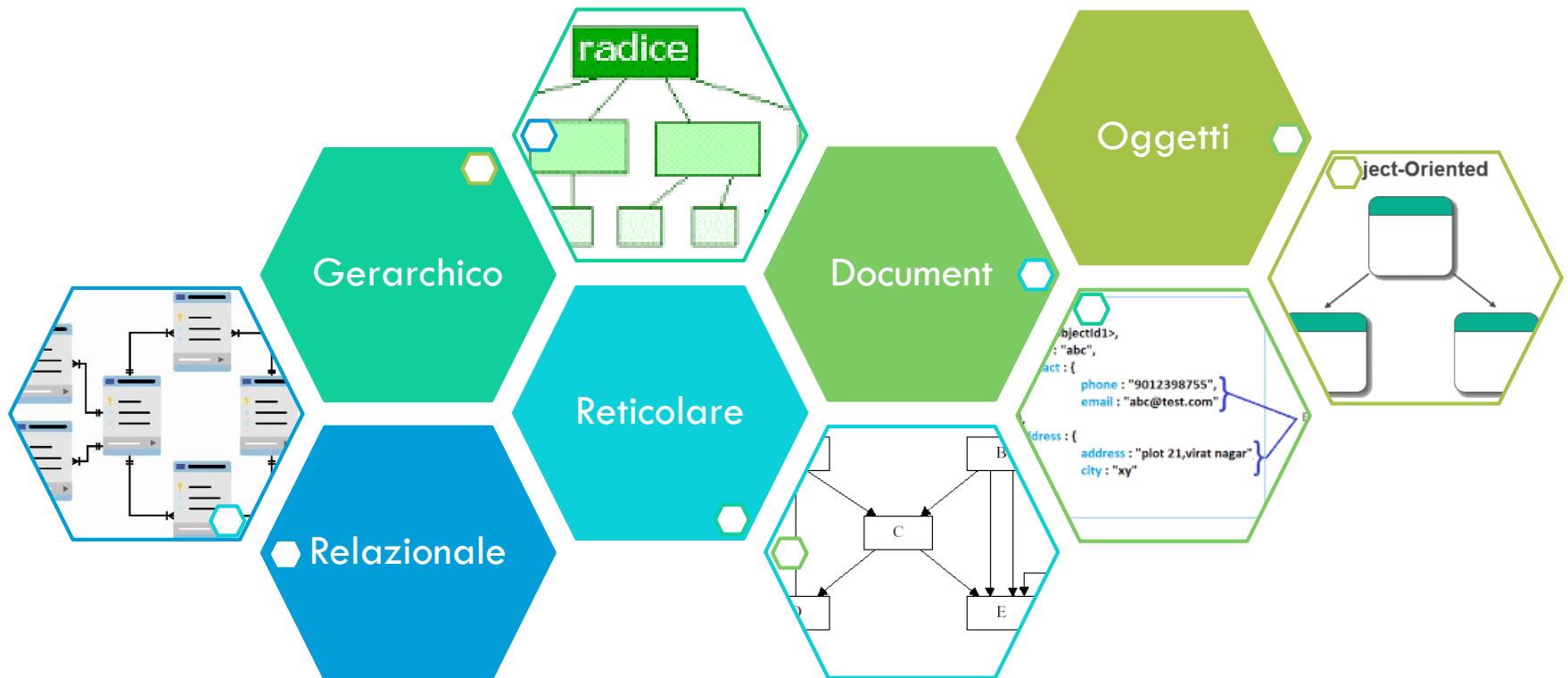
# Principali funzionalità di un DBMS

- Le caratteristiche fondamentali di un DBMS possono essere così riassunte:
  - ▣ supporto per almeno un **modello dei dati**;
  - ▣ uso di **cataloghi** per memorizzare la descrizione (schemi) di DB;
  - ▣ supporto di **viste multiple** sui dati condivisi tra più utenti;
  - ▣ **indipendenza** tra programmi e dati, e tra programmi e operazioni;
  - ▣ **gestione efficiente ed efficace** di grandi quantità di dati **persistenti e condivisi**;
  - ▣ **linguaggi di alto livello** per la definizione dei dati, l'interazione con il DB, e l'amministrazione e il controllo dei dati;
  - ▣ **funzionalità di supporto** per semplificare la descrizione delle informazioni, lo sviluppo delle applicazioni, l'amministrazione di un DB, le interfacce, ecc.
- Persistenza e condivisione richiedono meccanismi per garantire l'**affidabilità dei dati** (**fault tolerance**), per il **controllo degli accessi** e per il **controllo della concorrenza**.

# Il modello logico dei dati

- Dal punto di vista utente un DB è visto come una collezione di dati che modellano una certa porzione della realtà di interesse.
- L'**astrazione logica** con cui i dati sono resi disponibili all'utente definisce un **modello dei dati**; più precisamente:
  - un **modello dei dati** è una collezione di concetti utilizzati per descrivere i dati, le loro associazioni, e i vincoli che questi devono rispettare.
- Un ruolo di primaria importanza nella definizione di un modello dei dati è svolto dai **meccanismi che possono essere usati per strutturare i dati**.
  - Ad esempio esistono modelli in cui i dati sono descritti (solo) rispettivamente sotto forma di alberi (**modello gerarchico**), di grafi (**modello reticolare**), di oggetti complessi (**modello a oggetti**), di documenti XML, di documenti JSON o BSON, di coppie Key-Value, ecc.

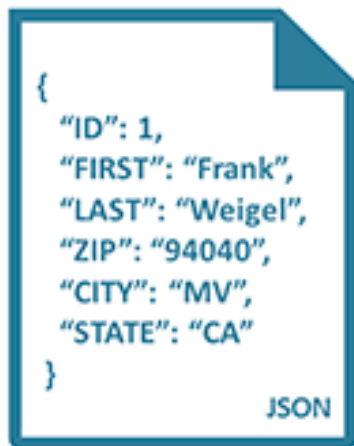
# I modelli logici



# Flessibilità del modello dati: un esempio

- I sistemi **NoSQL** usano modelli di dati molto differenti rispetto al modello relazionale, in quanto il principale obiettivo è la **flessibilità**.
- In un sistema orientato alla gestione di documenti, le informazioni sono memorizzate utilizzando un particolare formato, ad esempio **JSON**.
- L'aggregazione delle informazioni in un unico documento comporta certamente ridondanza, e problemi di inconsistenza, ma le prestazioni aumentano e ciò è molto importante, ad esempio, per applicazioni web.

## Document data model



## Relational data model



# Relational vs Document data model (1)

Esempio di schema di un DB relazionale per la gestione di post.

TAGS

<u>ID</u>	NAME
-----------	------

POSTS\_TAGS

<u>POST_ID</u>	<u>TAG_ID</u>
----------------	---------------

POSTS

<u>ID</u>	TITLE	SLUG	BODY	PUBLISHED	CREATED	UPDATED
-----------	-------	------	------	-----------	---------	---------

COMMENTS

<u>ID</u>	POST_ID	AUTHOR	EMAIL	BODY	CREATED
-----------	---------	--------	-------	------	---------

# Relational vs Document data model (2)

Schema

POSTS

ID

TITLE

SLUG

BODY

PUBLISHED

CREATED

UPDATED

COMMENTS

AUTHOR

EMAIL

BODY

CREATED

TAGS

- Lo schema in MongoDB (DBMS NoSQL) per la gestione di post prevede:
  - ▣ **embedding**: nidificazione di oggetti e array all'interno di un documento;
  - ▣ **linking**: realizzazione di riferimenti tra documenti.

```
{
  "_id" : ObjectId("4c03e856e258c2701930c091"),
  "title" : "welcome to MongoDB",
  "slug" : "welcome-to-mongodb",
  "body" : "Today, we're gonna totally rock your world...",
  "published" : true,
  "created" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)",
  "updated" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)",
  "comments" : [
    {
      "author" : "Bob",
      "email" : "bob@example.com",
      "body" : "My mind has been totally blown!",
      "created" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)"
    }
  ],
  "tags" : [
    "databases", "MongoDB", "awesome"
  ]
}
```

Esempio di documento BSON (Binary JSON)



# Natura autodescrittiva di una base di dati

- Il sistema di basi di dati contiene non solo la base di dati, ma anche una definizione o **descrizione completa della sua struttura** e dei suoi vincoli.
- Tale definizione è memorizzata nel **catalogo** del sistema, che contiene informazioni come la **struttura** di ciascun file, il **tipo** e il **formato** di memorizzazione di ogni dato e vari **vincoli** sui dati.
- Le informazioni memorizzate nel catalogo sono dette **metadati**.
- Il catalogo è usato dal DBMS e dagli utenti della base di dati che necessitano di informazioni sulla struttura della stessa.

# Catalogo

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

fqs

northwind

piscina

Tables

abbonamento

corsia

corso

fascia\_oraria

iscritto

iscrizione

istruttore

lezione

tipo\_abbonamento

turno

Views

Stored Procedures

Functions

sakila

sys

world

SQL File 6\* facequalityscores facequalityscores piscina piscina x

Info Tables Columns Indexes Triggers Views Stored Procedures Functions Grants Events

Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	Max Data Length	Index L
abbonamento	InnoDB	10	Dynamic	0	0	16.0 KIB	0.0 bytes	
corsia	InnoDB	10	Dynamic	0	0	16.0 KIB	0.0 bytes	
corso	InnoDB	10	Dynamic	0	0	16.0 KIB	0.0 bytes	
fascia_oraria	InnoDB	10	Dynamic	2	8192	16.0 KIB	0.0 bytes	
iscritto	InnoDB	10	Dynamic	0	0	16.0 KIB	0.0 bytes	

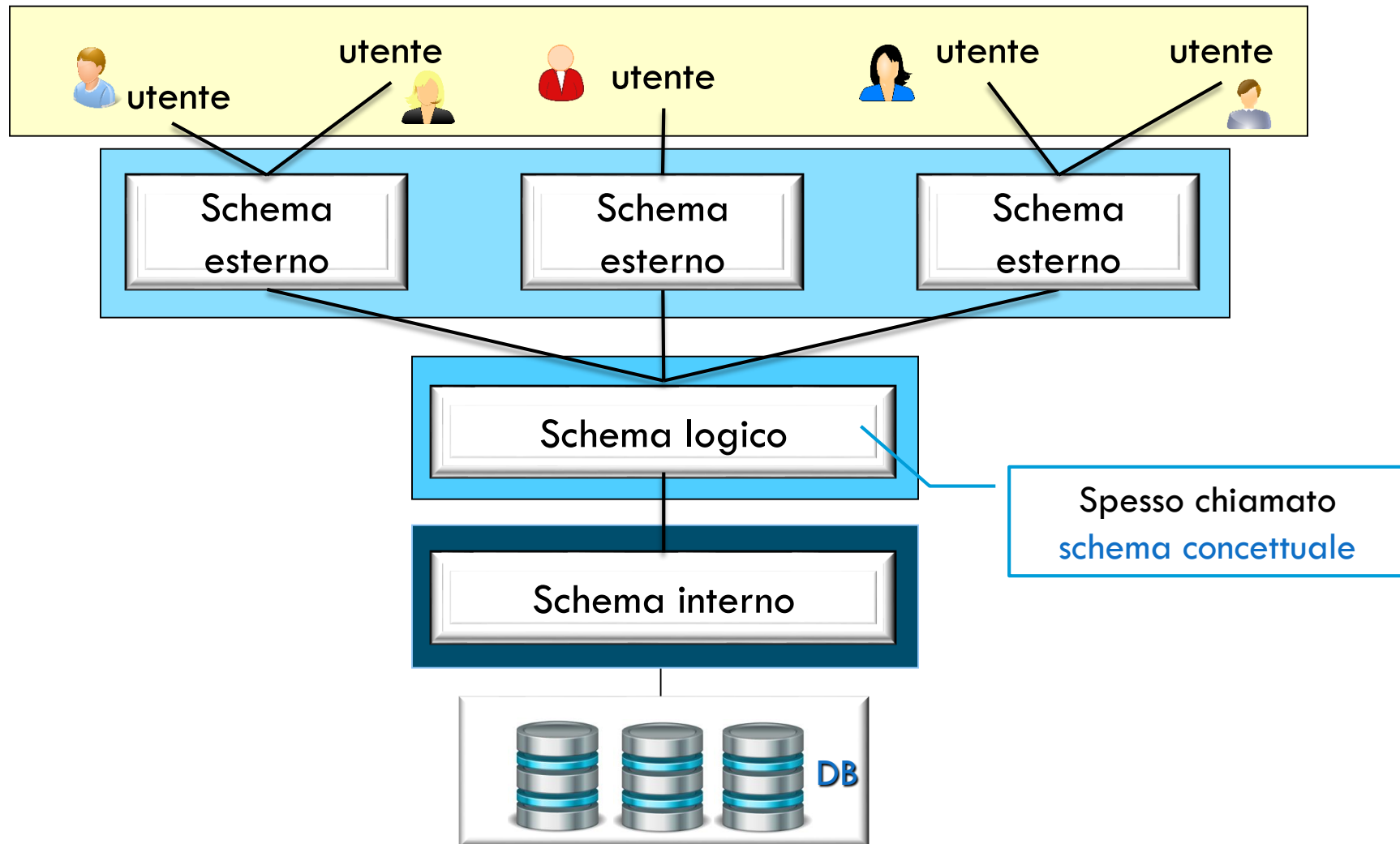
Table	Column	Type	Default Value	Nullable	Character Set	Collation	Priv
abbonamento	numero	int(11)		NO			sel
abbonamento	dataInizioValidita	date		NO			sel
abbonamento	dataPagamento	date		NO			sel
abbonamento	CF	char(16)		NO	utf8mb4	utf8mb4_0900_...	sel
abbonamento	idFascia	int(11)		NO			sel
abbonamento	durata	int(11)		NO			sel
corsia	vasca	int(11)		NO			sel
corsia	corsia	int(11)		NO			sel
corso	codCorso	int(11)		NO			sel
corso	nome	varchar(20)		NO	utf8mb4	utf8mb4_0900_...	sel
corso	descrizione	varchar(200)		NO	utf8mb4	utf8mb4_0900_...	sel
corso	durataLezioni	int(11)		NO			sel
corso	prezzo	float		NO			sel
		varchar(10)		YES	utf8mb4	utf8mb4_0900_...	sel
		char(1)		NO	utf8mb4	utf8mb4_0900_...	sel
		char(16)		NO	utf8mb4	utf8mb4_0900_...	sel
		int(11)		NO			sel
		time		NO			sel
		time		NO			sel
		char(16)		NO	utf8mb4	utf8mb4_0900_...	sel
		varchar(20)		NO	utf8mb4	utf8mb4_0900_...	sel
		varchar(20)		NO	utf8mb4	utf8mb4_0900_...	sel
		varchar(50)		NO	utf8mb4	utf8mb4_0900_...	sel

Table	Name	Unique	Index...	Index Comment
abbonamento	PRIMARY	Yes	BTREE	
abbonamento	ID_ABBONAMENTO_IND	Yes	BTREE	
abbonamento	FKsottoscrizione_IND	No	BTREE	
abbonamento	FKtipologia_IND	No	BTREE	
abbonamento	FKtipologia_IND	No	BTREE	
corsia	PRIMARY	Yes	BTREE	
corsia	PRIMARY	Yes	BTREE	
corsia	ID_CORSIA_IND	Yes	BTREE	
corsia	ID_CORSIA_IND	Yes	BTREE	
corso	PRIMARY	Yes	BTREE	
corso	ID_CORSO_IND	Yes	BTREE	

# Indipendenza fisica e logica

- Un importante obiettivo di un DBMS consiste nel fornire caratteristiche di **indipendenza logica** e **indipendenza fisica**. Queste caratteristiche sono soddisfatte in gran parte dai sistemi relazionali.
- **Indipendenza fisica**
  - L'organizzazione fisica dei dati dipende da considerazioni legate all'efficienza delle strutture dati adottate; la riorganizzazione fisica dei dati, o la creazione di strutture d'accesso aggiuntive, non deve comportare modifiche allo schema logico del DB - lasciando inalterate anche le particolari viste d'utente- né deve causare effetti collaterali sui programmi applicativi.
- **Indipendenza logica**
  - Pur in presenza di uno schema logico integrato non è spesso utile o conveniente che ogni utente ne abbia una visione uniforme; è desiderabile che modifiche allo schema logico non comportino aggiornamenti degli schemi esterni e delle applicazioni.
- La soluzione porta a quella che è comunemente nota come **architettura a 3 livelli** di un DBMS, supportata da software di mapping tra i diversi livelli.

# Architettura a tre livelli di un DBMS



# Il livello fisico (o interno)

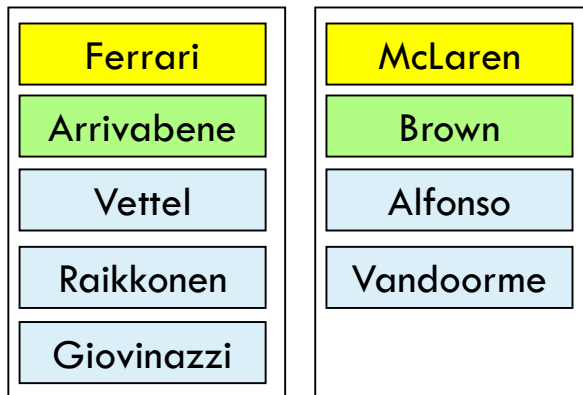
- Il “**DB fisico**” consiste di una serie di file, residenti su dispositivi di memoria permanenti che contengono dati, indici e altre tipologie di strutture.
- Lo **schema fisico** descrive come il DB, definito a livello logico, è rappresentato a livello fisico (ad esempio: in quale/i file è memorizzata una relazione, quali cammini d'accesso sono disponibili, ecc.).

SCUDERIE

<u>Scuderia</u>	TeamPrincipal
Ferrari	Arrivabene
McLaren	Brown

Schema logico

Schema fisico



PILOTI\_SCUDERIE

<u>Scuderia</u>	<u>Pilota</u>
Ferrari	Vettel
Ferrari	Raikkonen
McLaren	Alfonso
McLaren	Vandoorme
Ferrari	Giovinazzi

La gestione del **DB fisico** è a carico del **DBA (Data Base Administrator)** e non degli utenti, i quali possono quindi concentrarsi su aspetti di più alto livello.

File: C:\Users\DB\_CORSE\_AUTO\Data\Scuderie

# Il livello delle viste (o esterno)

- Il **livello esterno** è costruito a partire dallo schema logico integrato mediante la definizione di viste ad hoc che descrivono parte dello schema logico secondo le esigenze di operatività dei diversi utenti.
  - Ad esempio è possibile definire una vista che combina i dati di più relazioni:

TEAMPRINCIPAL\_PILOTI

<u>TeamPrincipal</u>	<u>Pilota</u>
Arrivabene	Vettel
Arrivabene	Raikkonen
Arrivabene	Giovinazzi
Brown	Alfonso
Brown	Vandoorme

La distinzione tra livello esterno e logico può, in molti casi, risultare trasparente agli utenti, che, ad esempio, in un RDBMS “vedono” semplicemente un insieme di tabelle.

# Utilità delle viste

- Oltre a fornire una visione “**personalizzata**” del DB, le viste possono svolgere un ruolo importante anche per diversi altri motivi:
  - una **ristrutturazione dello schema integrato** può, in alcuni casi, essere opportunamente “mascherata” facendo uso di viste;
  - mediante le viste è possibile regolare meglio il **controllo degli accessi** al DB, ad es. mascherando dati riservati;
  - le viste possono essere usate per “**calcolare dinamicamente**” nuovi dati a partire da quelli memorizzati nel DB, senza per questo introdurre ridondanza.

Pilota	DataNascita
Vettel	03/07/1987
Raikkonen	17/10/1979

Il 14/01/2018



Pilota	Età
Vettel	30
Raikkonen	38

# Condivisione: regolamentare gli accessi

- Gli utenti di un DB sono naturalmente classificabili in diverse tipologie, a cui vanno pertanto associate **autorizzazioni** distinte, ad esempio:
  - uno **studente** può leggere i propri dati, ma non quelli di altri studenti; inoltre non può modificare l'elenco degli esami sostenuti;
  - un **docente** può leggere i dati dei soli studenti del proprio corso; non può modificare l'elenco degli esami già sostenuti da uno studente, ma può registrare esami del proprio corso;
  - la **segreteria studenti** può leggere i dati di tutti e può registrare nuovi studenti.
- La gestione delle autorizzazioni può essere oltremodo complessa, per questo motivo sono previste specifiche figure di **Data Base Administrator** che conferiscono agli utenti i “giusti” privilegi.
- Il **DCL** di **SQL** semplifica notevolmente la concessione dei privilegi (ad esempio leggere o modificare i dati di una tabella) a una classe di utenti; inoltre un utente può concedere a sua volta privilegi ad altri utenti.



# Concorrenza e protezione da guasti

- Un DBMS deve garantire che gli accessi ai dati, da parte di diverse applicazioni, non interferiscano tra loro violando vincoli di integrità e alterando la consistenza del DB.
- A tale scopo è pertanto necessario far ricorso a opportuni meccanismi di **controllo della concorrenza**.
- Quando un'applicazione deve effettuare più operazioni di modifica, è possibile che per qualche motivo (disk failure, interruzioni di rete, intervento dell'utente, transaction abort, errori software, ...) solo una parte di queste sia effettivamente eseguita.
- In questo caso, per garantire l'integrità e la consistenza dei dati, il DBMS deve provvedere ad **annullare** tali modifiche.
- Una transazione è vista come una sequenza di operazioni elementari di **lettura (R)** e **scrittura (W)** di oggetti del DB che, a partire da uno stato iniziale consistente del DB, porta il DB in un nuovo stato finale consistente.

# RDBMS: Proprietà ACID

**A** TOMICITY: la transazione è indivisibile nella sua esecuzione e la sua esecuzione deve essere **o totale o nulla**.

**C**ONSISTENCY: quando inizia una transazione il database si trova in uno stato consistente e al termine della transazione il database deve ancora essere in un stato consistente (**nessun vincolo sui dati deve essere violato**).

**I**SOLATION: ogni transazione deve essere **eseguita in modo isolato** e indipendente dalle altre transazioni, l'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione. In altre parole un'esecuzione concorrente di più transazioni è “equivalente” a un'esecuzione seriale delle stesse.

**D**URABILITY: detta anche **persistenza**; al termine della transazione i dati modificati devono essere stati memorizzati in un dispositivo durevole (es. hard disk) e deve essere registrato il completamento della transazione stessa.

# I linguaggi dei DBMS

- Un DBMS mette a disposizione diversi linguaggi per interagire con le BD. Il livello di astrazione dei linguaggi offerti dipende fortemente dal modello dei dati a cui ci si riferisce.
- Una comune distinzione classifica i linguaggi sulla base delle funzioni svolte:
  - ▣ DDL (Data Definition Language)
    - per definire gli schemi (logici, esterni, interni)
  - ▣ DML (Data Manipulation Language)
    - per interrogare e modificare un DB
  - ▣ DCL (Data Control Language)
    - include comandi di vario tipo, ad es. per il controllo degli accessi

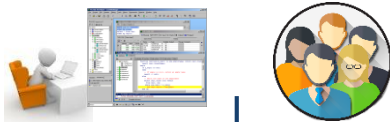
Il linguaggio SQL nei RDBMS riunisce in sé istruzioni delle suddette tre tipologie.

# Linguaggi per lo sviluppo di applicazioni

- Oltre alle istruzioni necessarie per la manipolazione della base di dati, per realizzare applicazioni è necessario disporre di linguaggi general purpose. In generale, con riferimento particolare ai RDBMS, sono disponibili diverse alternative, tra cui:
  - **Linguaggio nativo**
    - Il DBMS dispone di un linguaggio Turing-completo (esempi: **Oracle PL/SQL**, **Postgres PLpg/SQL**).
  - **Linguaggio ospite con procedure esterne**
    - Le istruzioni per manipolare i dati vengono invocate all'interno di un linguaggio convenzionale (esempi: uso di **JDBC per Java**, **ODBC per altri linguaggi di programmazione**).
  - **Linguaggio ospite con precompilatore**
    - “Embedded statements” in un linguaggio convenzionale che richiedono la presenza di un precompilatore (esempio: **PRO\*C per SQL in ambiente Oracle**).

# Principali moduli di un DBMS

Users & applications



## Query manager

- analizza, autorizza, ottimizza ed esegue le richieste

## Storage manager

- gestisce i dispositivi di storage

## Transaction manager

- coordina l'esecuzione delle transazioni

## Logging & Recovery Manager

- garantisce **Atomicity** e **Durability**, protegge dai malfunzionamenti

DBA



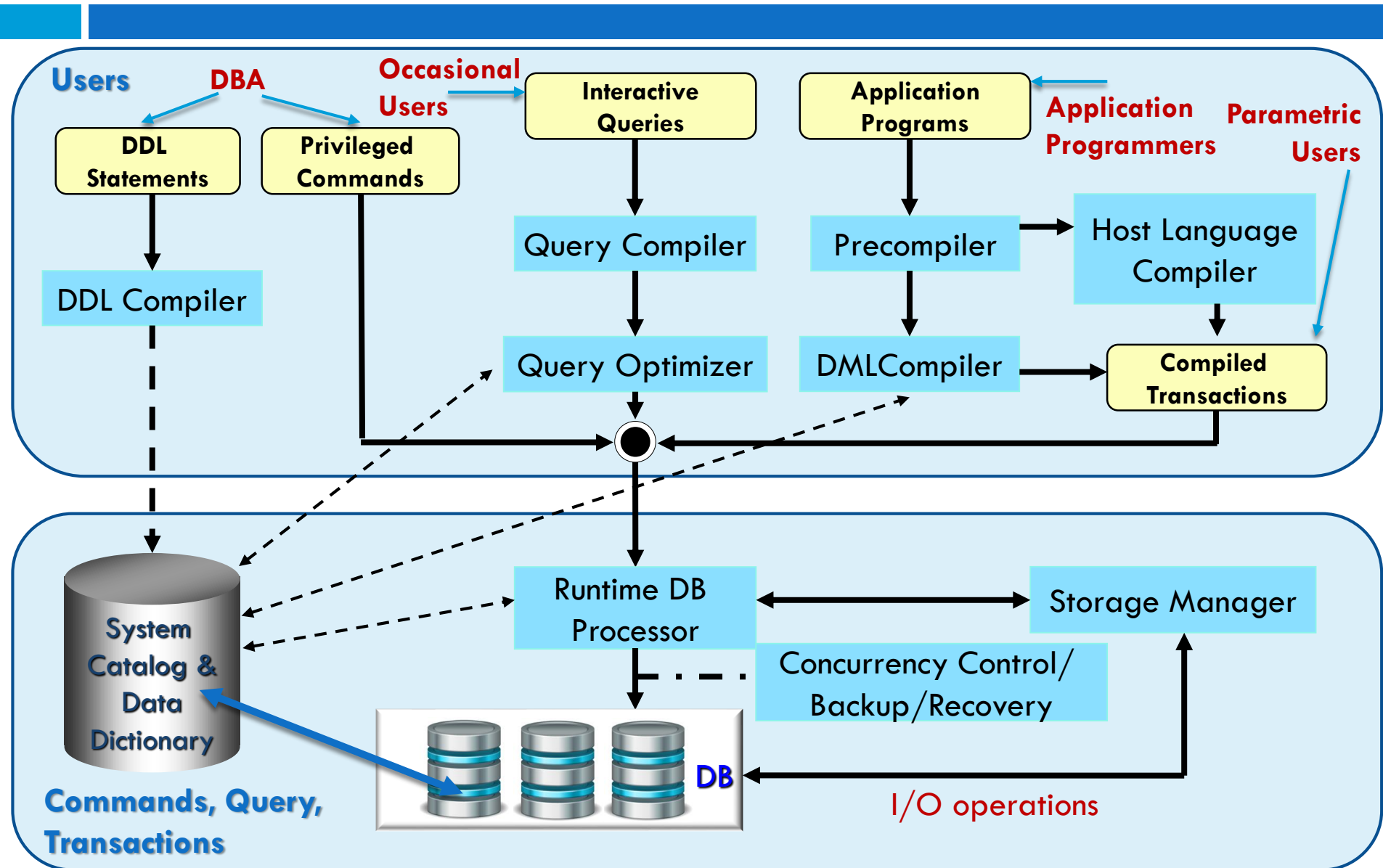
## DDL compiler

- genera parte dei controlli per garantire **Consistency**

## Concurrency Manager

- gestisce accessi concorrenti e garantisce **Isolation**

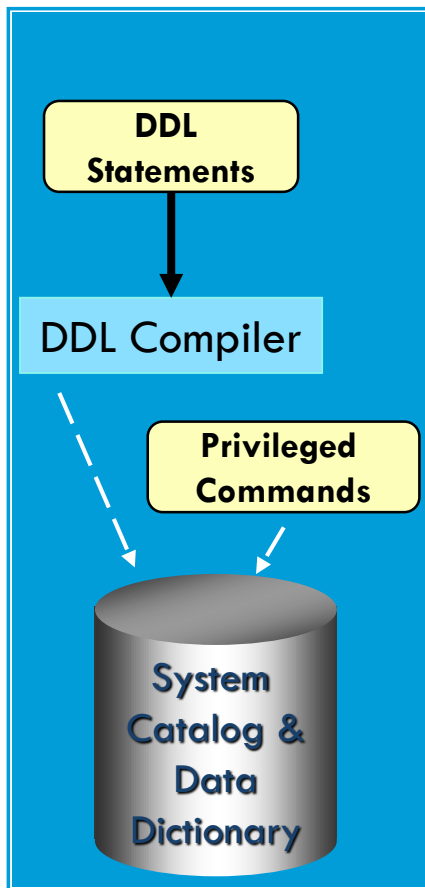
# Moduli di un RDBMS (1)



# Moduli di un RDBMS (2)

- La base di dati e il catalogo del DBMS sono solitamente memorizzati su disco.
- L'accesso al disco è controllato essenzialmente dal sistema operativo che ne pianifica le operazioni di lettura e scrittura.
- Numerosi DBMS possiedono il proprio modulo di gestione del buffer per pianificare la operazioni di lettura e scrittura su disco.
- Un gestore dei dati archiviati, modulo DBMS di alto livello, controlla l'accesso alle informazioni memorizzate su disco (database e catalogo).

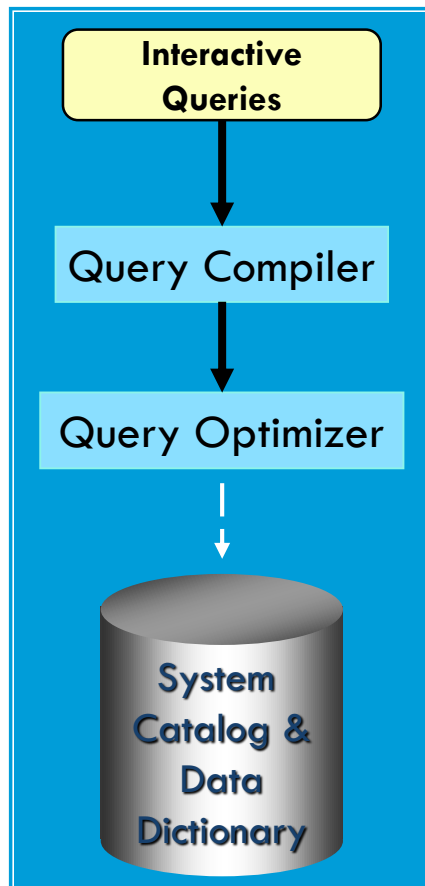
# Utenti: DBA



- Lo staff di amministrazione lavora sulla **definizione della base di dati** e della sua **manutenzione**, modificando le definizioni per mezzo del **linguaggio DDL** e di altri **comandi privilegiati**.
- Il **compilatore DDL** elabora definizioni di schema (nel DDL) e memorizza descrizioni degli schemi (meta-dati) nel catalogo del DBMS.

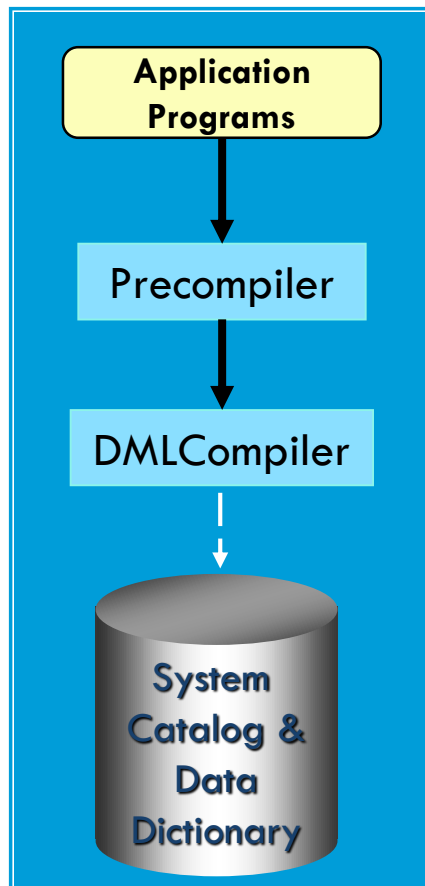


# Utenti: occasional users



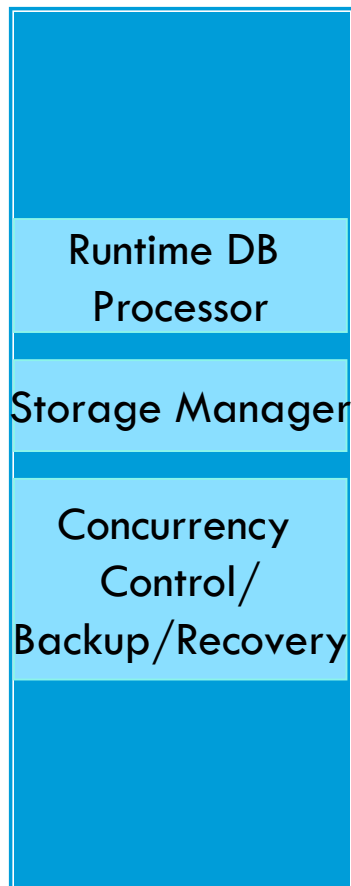
- Gli utenti occasionali utilizzano le cosiddette **interfacce interattive** (interactive queries). Non utilizzano applicazioni che possono generare le interrogazioni in modo automatico.
- Queste interrogazioni sono trasformate in una forma interna dal **compilatore delle interrogazioni** che le analizza e ne verifica la correttezza (sintassi, nomi, ecc.)
- L'interrogazione interna viene poi ottimizzata dall'**ottimizzatore delle interrogazioni** che si occupa di riorganizzare le operazioni, eliminare le ridondanze, utilizzare gli opportuni algoritmi per l'esecuzione.
- L'ottimizzatore consulta il **catalogo** per avere informazioni **statistiche** e informazioni relative alla **memorizzazione fisica dei dati** e genera il codice eseguibile.

# Utenti: application programmers



- I programmatori di applicazioni scrivono programmi in **linguaggi ospite** che vengono quindi passati a un pre-compilatore che estrae **comandi DML**.
- I comandi sono inviati al **compilatore DML** per la compilazione in codice oggetto.
- Il resto del programma è inviato al **compilatore del linguaggio ospite**.
- Il codice oggetto per i comandi DML e quello per il resto del programma sono collegati (**linked**), portando alla creazione di un **codice eseguibile** che include chiamate al processore di esecuzione della base di dati.

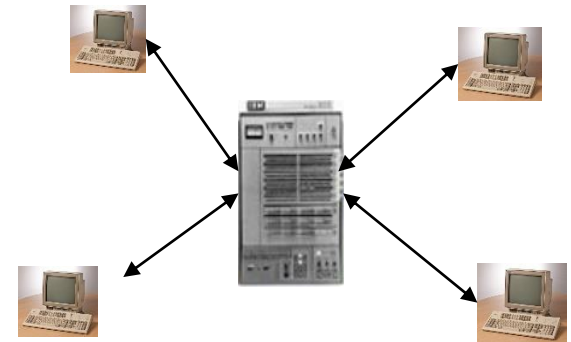
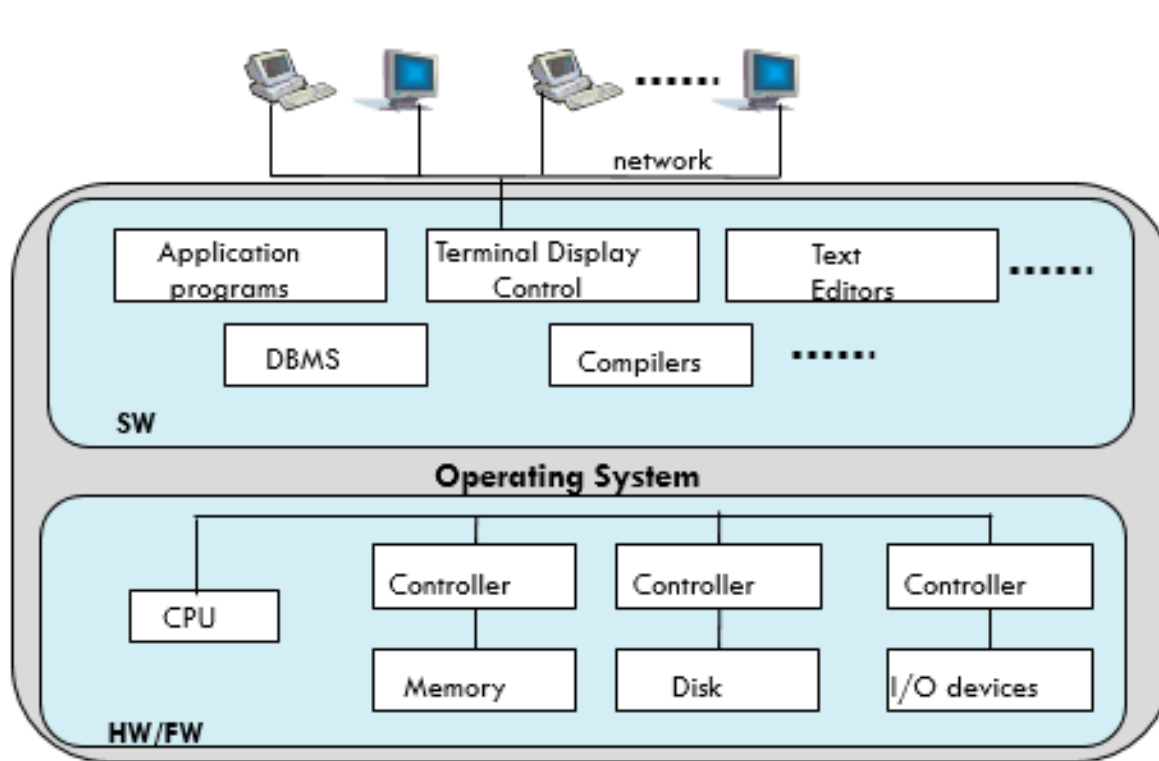
# Moduli DBMS: comandi, query, transazioni



- Il **processore runtime** della base di dati si occupa di eseguire:
  - ▣ i **comandi privilegiati**;
  - ▣ i **piani di esecuzione** delle interrogazioni;
  - ▣ le **transazioni parametriche** predefinite con specifici parametri
- Il processore interagisce con il **catalogo** per aggiornare le statistiche.
- Il processore interagisce con il **gestore dei dati memorizzati**, il quale a sua volta usa i servizi del **sistema operativo** per realizzare le operazioni in input e output di basso livello tra disco e memoria centrale.
- I sistemi di **backup/recovery** e di **controllo della concorrenza** sono integrati nel processore runtime per la gestione delle transazioni.

# Architettura DBMS centralizzata

- In un'architettura centralizzata tutte le funzionalità del DBMS, l'esecuzione delle applicazioni e la gestione dell'interfaccia utente sono eseguite su una singola macchina.

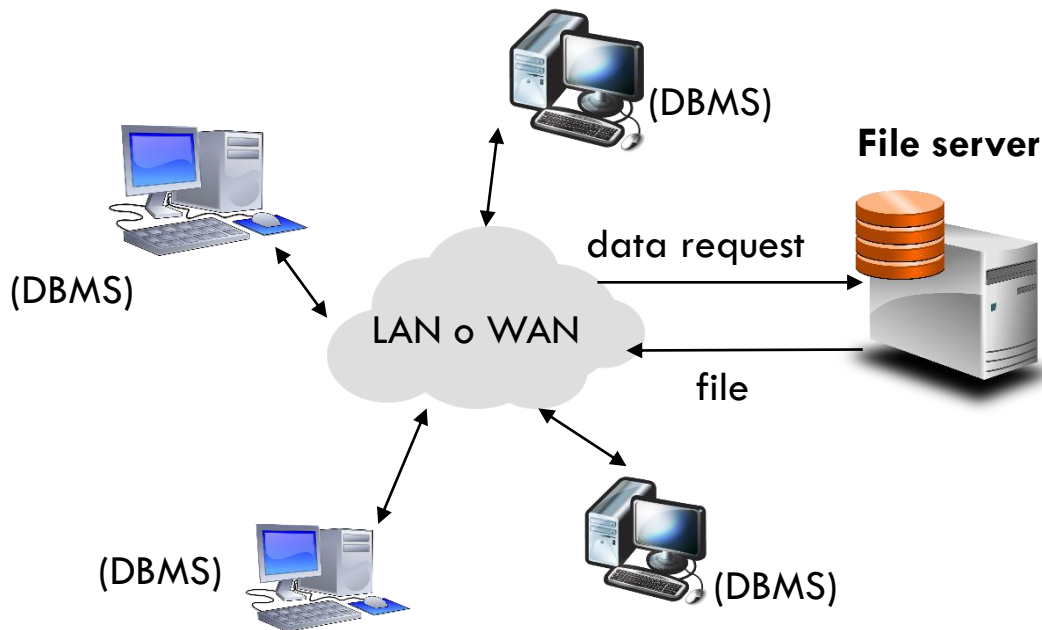


## Teleprocessing

Architettura tipica degli anni 70-80 con sistema centralizzato (minicomputer o **mainframe** nelle grandi organizzazioni) e terminali "stupidi".

# Architettura File Server

- Negli 90 si afferma la tendenza (**downsizing**) a sostituire costosi mainframe con reti di computer a basso costo (workstation, PC, server Unix). Si diffondono pertanto architetture **file-server** e **client-server**.



## Svantaggi:

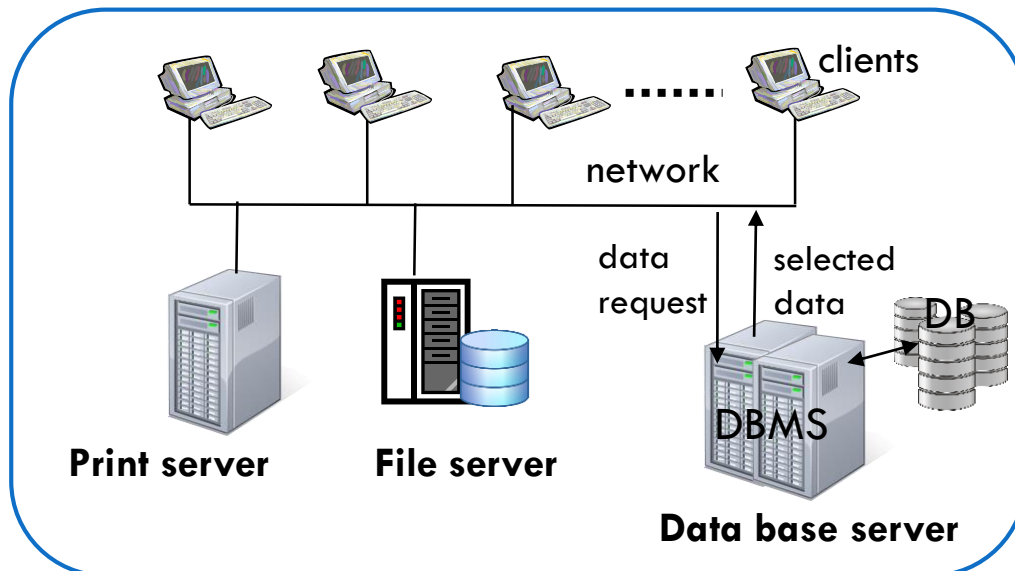
- elevati costi dei client (workstation);
- traffico dati elevato sulla rete;
- più DBMS accedono agli stessi dati e ciò comporta problemi di integrità, concorrenza e recovery.

Il file server agisce come gestore di **shared store**; l'elaborazione avviene nelle varie workstation dotate di DBMS e applicazioni.

# Architettura Client/Server

- L'architettura **client-server** è stata sviluppata per gestire in modo efficiente ambienti in cui sono presenti un gran numero di postazioni PC, workstation, file server, ecc.
- In questo contesto un **client** è tipicamente una macchina utente in grado di fornire funzionalità d'interfaccia e di elaborazione locale.
- Un **server** è un sistema in grado di fornire servizi di varia natura alle macchine client.

## Logical two-tier client-server architecture



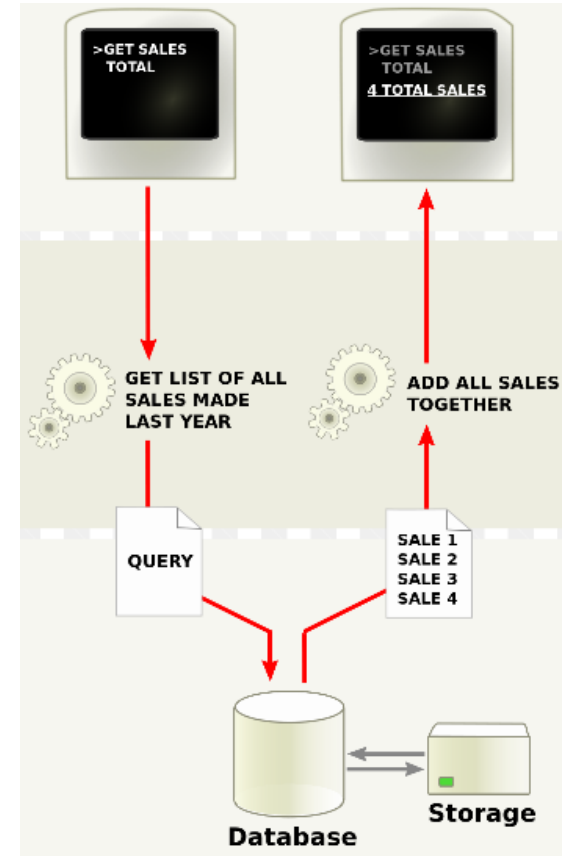
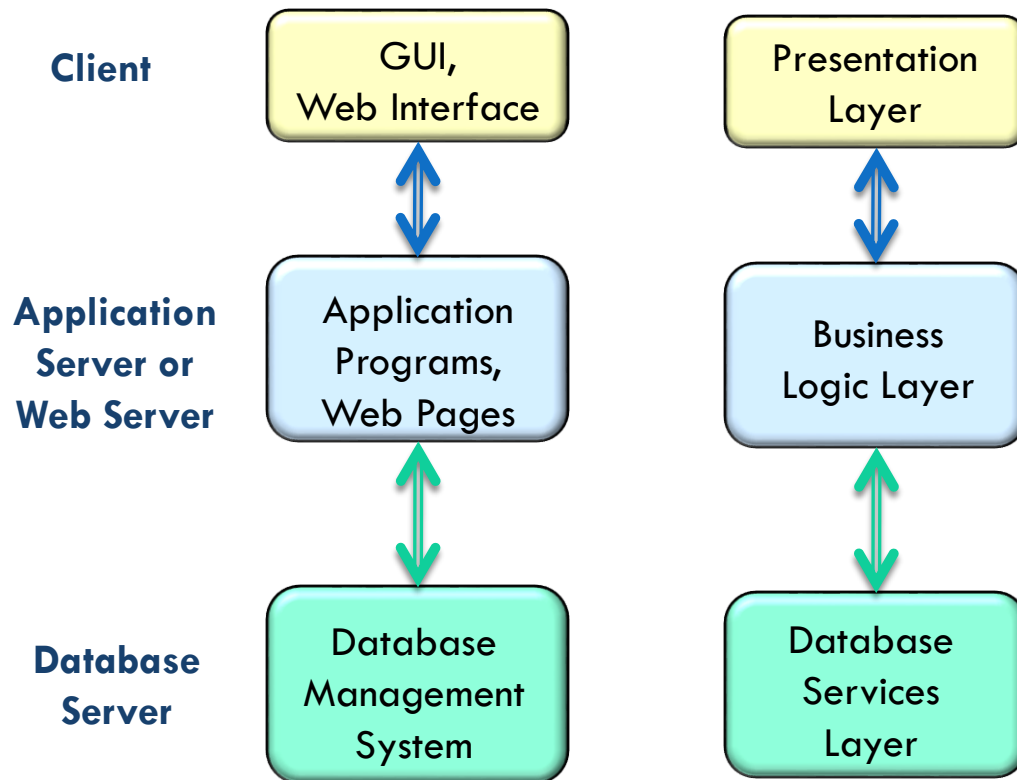
- Chiara separazione tra attività dei client e del server (DMBS);
- **fat (thick) client** o **thin client** in base al ruolo, più o meno pesante, da essi svolto nella logica applicativa;
- possibili diverse tipologie;
- vantaggi in termini di costi di hw, di volumi di traffico, di efficienza, di sicurezza e di consistenza dei dati.

# Two-tier vs Three-tier architecture

- In un'architettura **two-tier** (a due livelli) le componenti software sono distribuite su due sistemi: client e server. Tuttavia vi sono alcuni svantaggi:
  - scarsa **scalabilità** con centinaia-migliaia di utenti;
  - **amministrazione onerosa** dei client ad esempio per l'installazione di aggiornamenti alla logica applicativa o di nuove applicazioni;
  - thick client richiedono una configurazione ricca in termini di **risorse** (CPU, RAM, ...) per soddisfare esigenze di **efficienza** nell'esecuzione delle transazioni.
- La diffusione del Web ha modificato sostanzialmente i ruoli dei client e dei server, richiedendo il passaggio ad architetture a tre livelli (**three-tier architecture**) in cui è presente un livello intermedio (spesso chiamato **application server** o **web server**).
- Nell'architettura **three-tier** è possibile garantire un migliore grado di sicurezza rispetto agli accessi, infatti:
  - al database server si accede solo attraverso il livello intermedio;
  - i client non possono accedere direttamente al database server.

# Three-tier client-server architecture

Logical three-tier client/server architecture: due tipiche nomenclature



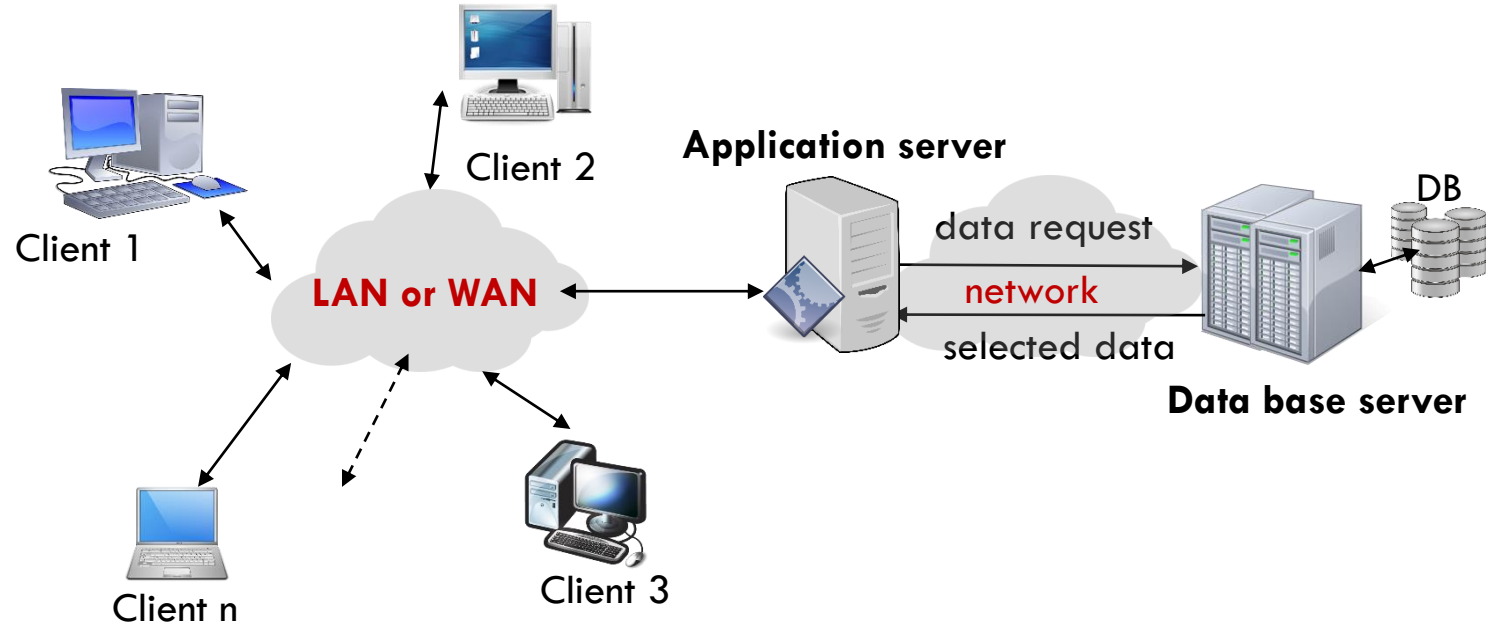
Fonte: Wikipedia

- ❑ **Presentation Layer:** si occupa dell'interfaccia con l'utente.
- ❑ **Business Logic Layer:** gestisce regole e vincoli intermedi prima di trasferire risultati all'utente o trasmettere dati al sottostante livello DBMS.
- ❑ **Database Services Layer:** è responsabile dei servizi di accesso al DB.



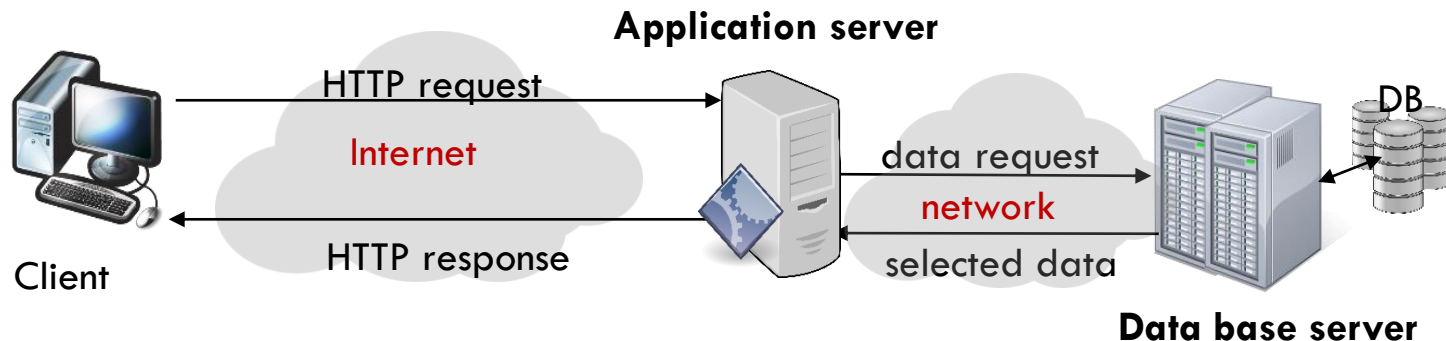
# Three-tier architecture: vantaggi

- Costi ridotti per thin client che richiedono minori risorse in termini di hw, ad esempio le applicazioni sono eseguite in ambiente web browser.
- La logica applicativa è centralizzata in un singolo application server;
- È garantita una maggiore modularità in quanto è più semplice sostituire un livello senza incidere sugli altri.
- È più semplice effettuare un bilanciamento del carico di lavoro grazie alla netta separazione tra logica applicativa e funzionalità database.



# WIS: Web Information System

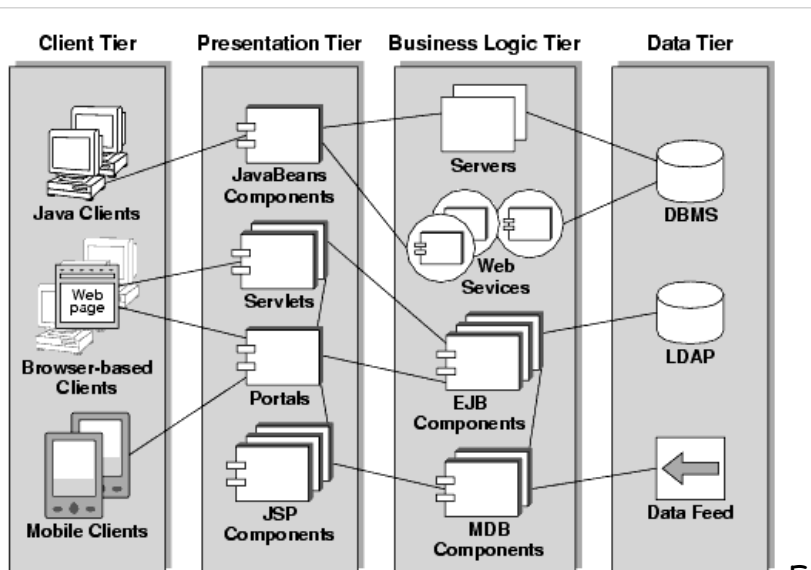
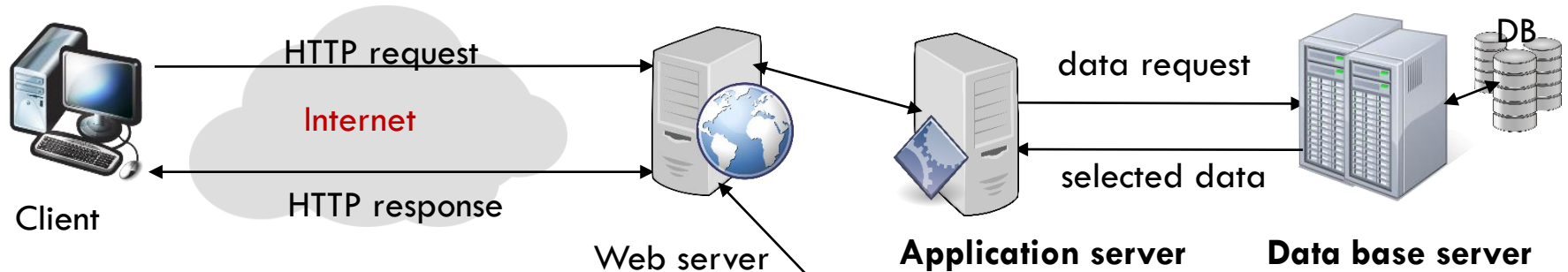
L'architettura **three-tier client-server** ben si presta a uno scenario in ambiente web, dove il browser svolge il ruolo di thin-client. Si parla in questo caso di **WIS** (**W**eb **I**nformation **S**ystem).



L'adozione di browser come thin-client riduce sostanzialmente i costi di sviluppo del software.

# N-tier architecture

Sono possibili **architetture N-tier**; ad esempio separando web server e application server in un **web-based information system**:



HTML pages

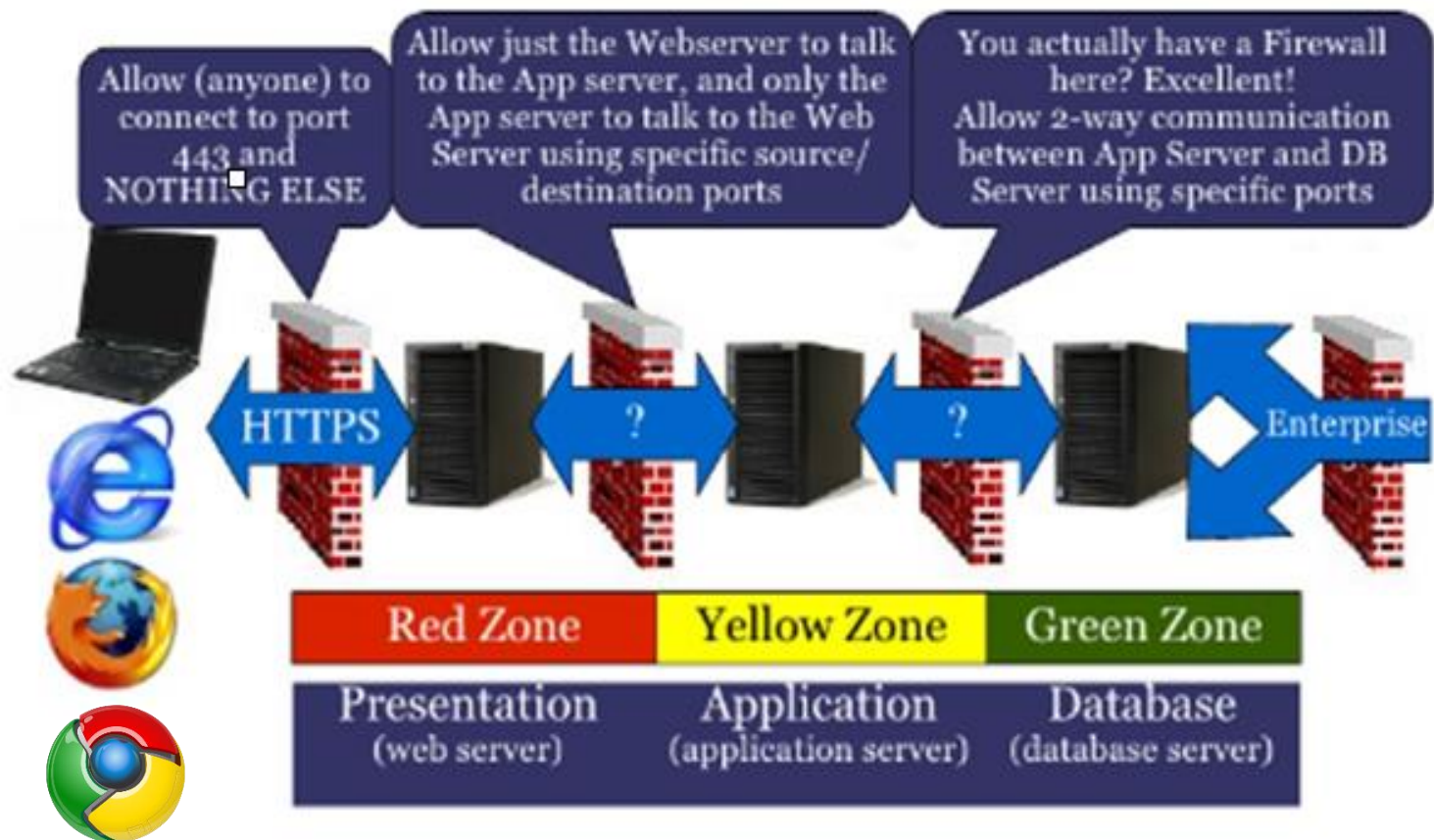
- Load balancing flessibile introducendo più web server
- Solo il contenuto dinamico è trasferito dall'application server, mentre il contenuto statico delle pagine HTML è gestito dal web server.

Un'implementazione 4-tier su **Java Platform Enterprise Edition**.

Fonte: <https://docs.oracle.com/cd/E19199-01/817-5085/concepts.html>

# Sicurezza degli accessi ai vari livelli

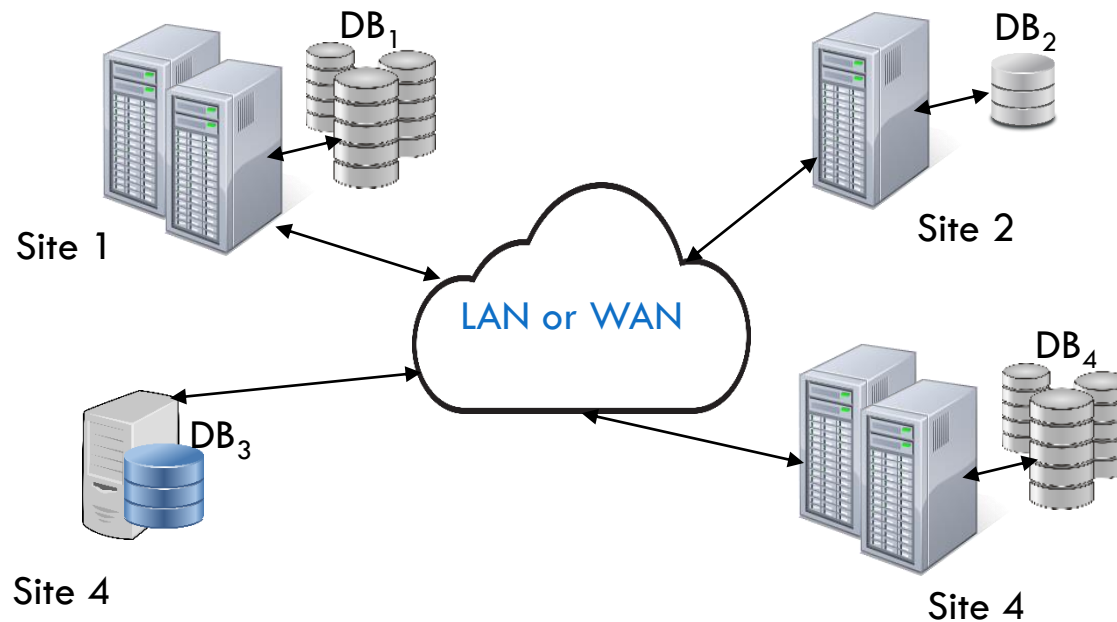
Nelle architetture viste si deve naturalmente porre molta attenzione alle problematiche di protezione da accessi non autorizzati. Ciò implica il ricorso a opportuni **sottosistemi firewall**.



Fonte: <http://www.slideshare.net/jikbal/web-application-security-with-php>

# Peer-to-Peer architecture

Nelle architetture **Peer-to-Peer** gli scambi di informazioni avvengono in assenza di un'autorità centrale, non vi sono né client né server dedicati e i vari siti possono dinamicamente creare nuove relazioni client-server.

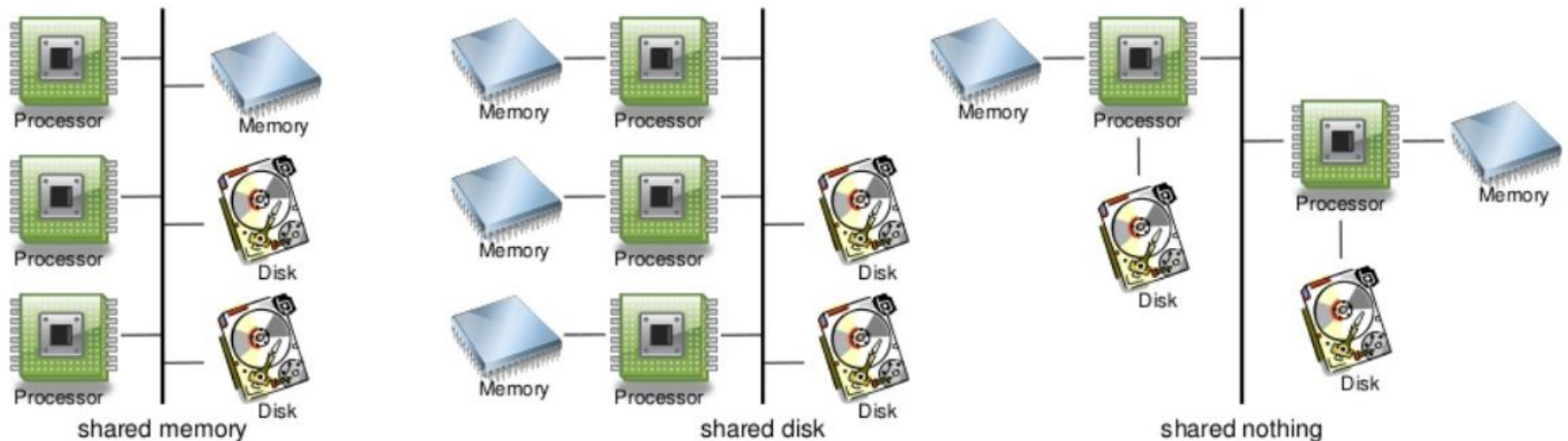


# DDBMS: cenni

- Un DB distribuito (DDB) è una collezione di dati che appartengono logicamente alla stessa organizzazione, e sono distribuiti su nodi di una rete di calcolatori.
- Un Distributed Data Base Management System (DDBMS) è il sistema software che permette la gestione di un DDB.
- Un DDB non è una meramente una “collezione di DB”, nel senso che tra i dati memorizzati sui diversi nodi (siti del DDB) deve esistere una **correlazione logica**; ciò si riflette nella presenza di “**applicazioni distribuite**” (o globali) che elaborano dati allocati su più nodi.
- Vi sono tre dimensioni implementative ortogonali:
  - livello di distribuzione;
  - autonomia;
  - eterogeneità.
- Diverse sono le possibili architetture:
  - client-server;
  - peer-to-peer;
  - multi-DBMS.

# Parallel DBMS: cenni

- Architetture che mirano a migliorare le prestazioni attraverso il ricorso al **parallelismo dell'esecuzione** di varie operazioni.
- Una classe di sistemi sfrutta **architetture multi-processore** e sono possibili nella sostanza tre diversi scenari come in figura.



Fonte: <http://www.slideshare.net/signer/dbms-architectures-and-features-3538218>

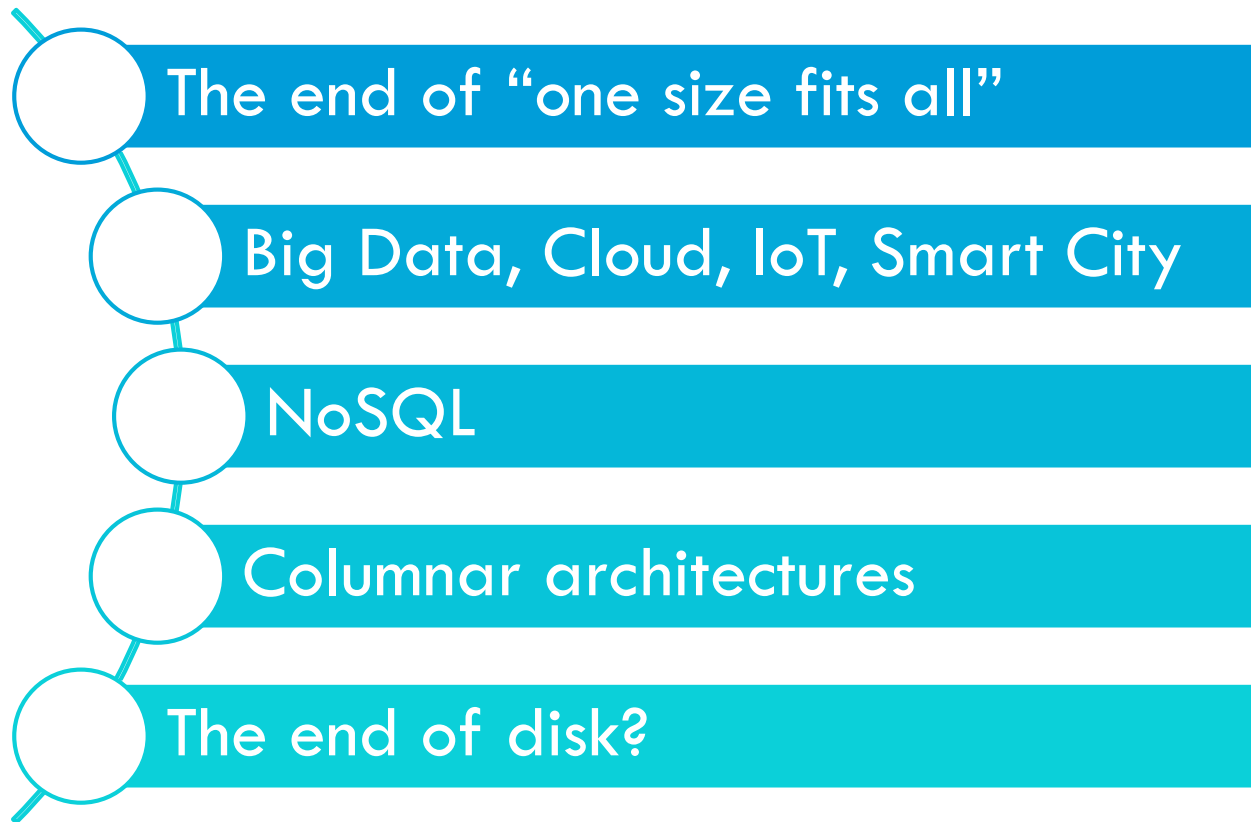
Un'altra classe di sistemi sfrutta **architetture ibride** (es. shared nothing + shared disk con un cluster di computer connessi in rete).

# Cloud database: cenni

- La più recente evoluzione delle architetture risiede nello sviluppo di piattaforme di **cloud computing**; si sono ormai affermate soluzioni cloud per la gestione di basi di dati ove l'accesso è fornito come servizio da parte di un provider. I servizi cloud si prendono cura di garantire una soddisfacente scalabilità dell'architettura hw/sw e un'elevata disponibilità della base di dati. Essi rendono lo stack del software sottostante trasparente all'utente.
- Sostanzialmente vi sono due modalità di gestione.
  - ▣ **Virtual machine image**: gli utenti possono acquistare istanze di macchine virtuali per un tempo limitato, dedicate alla gestione di database; essi possono anche caricare le proprie machine image con database già installato, oppure utilizzare una macchina già configurata con una versione ottimizzata di un database.
  - ▣ **Database-as-a-service (DBaaS)**: il fornitore del servizio si prende la responsabilità di installare e mantenere il database, e i proprietari delle applicazioni pagano un importo commisurato all'uso del servizio.



# Nuove tendenze



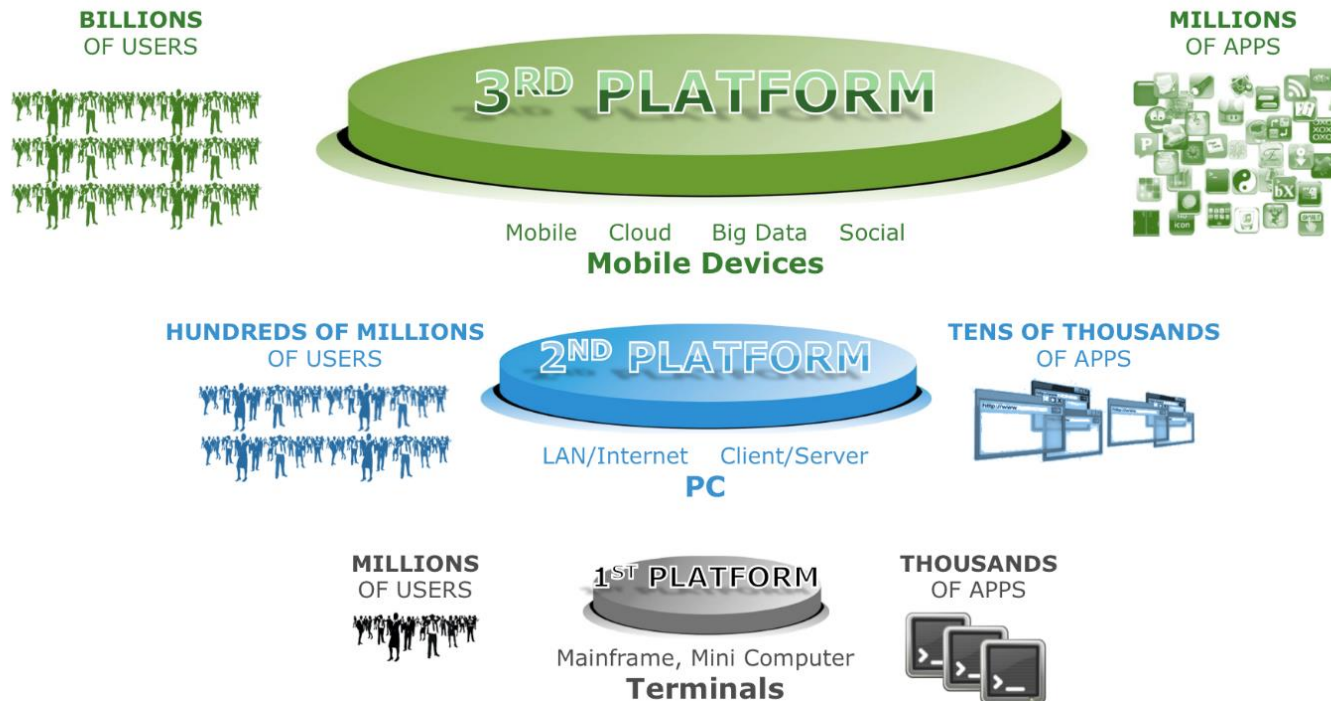
Top 5 Trends in Database Technology, Guy Harrison, Dell Software Group.

# The end of “one size fits all”

## Nuove esigenze

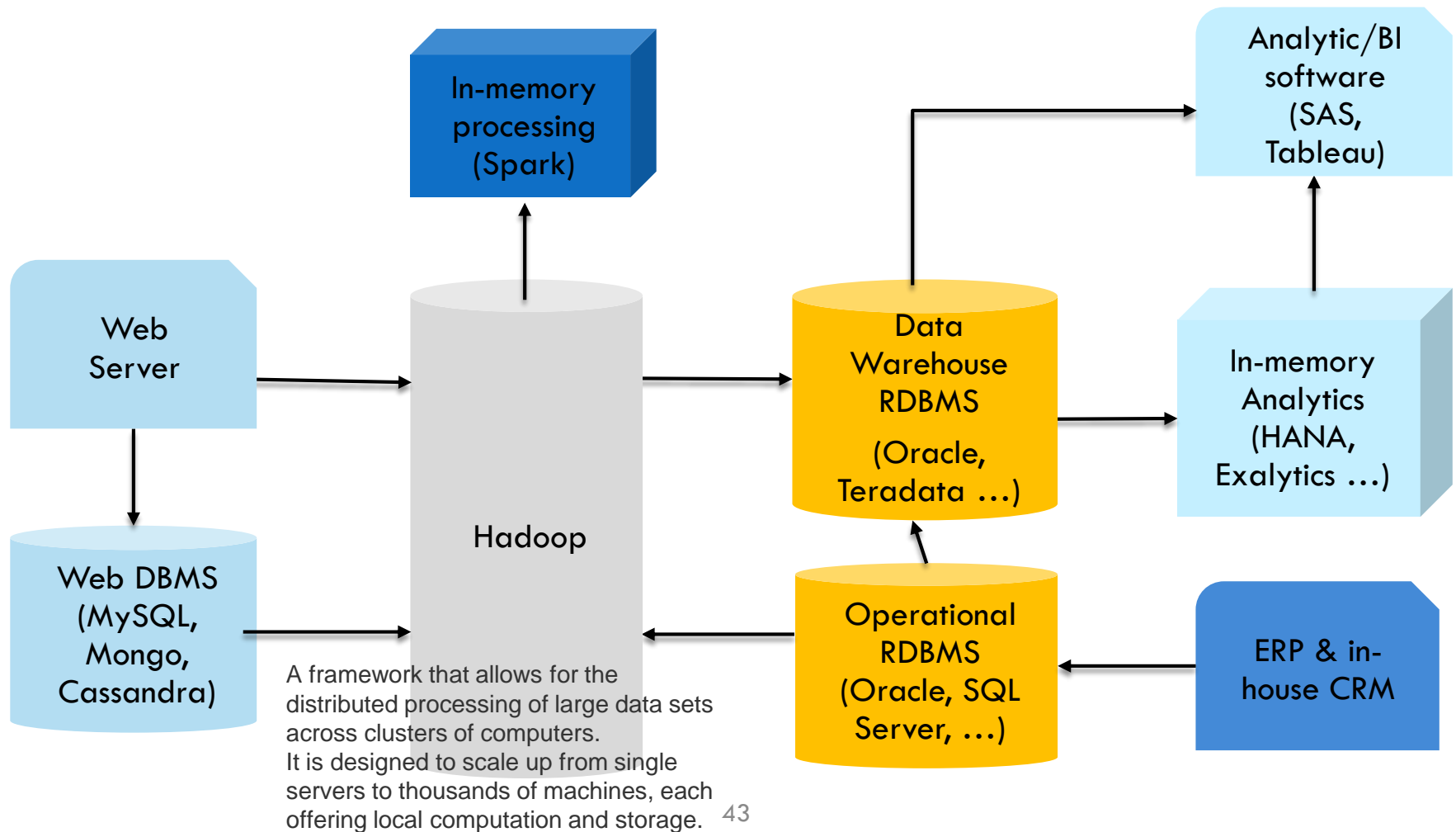
- ❑ Elevata disponibilità a livello globale
- ❑ Volumi rilevanti di dati
- ❑ Dati non strutturati
- ❑ Prestazioni
- ❑ ...

Una singola architettura non può soddisfare tutte queste esigenze

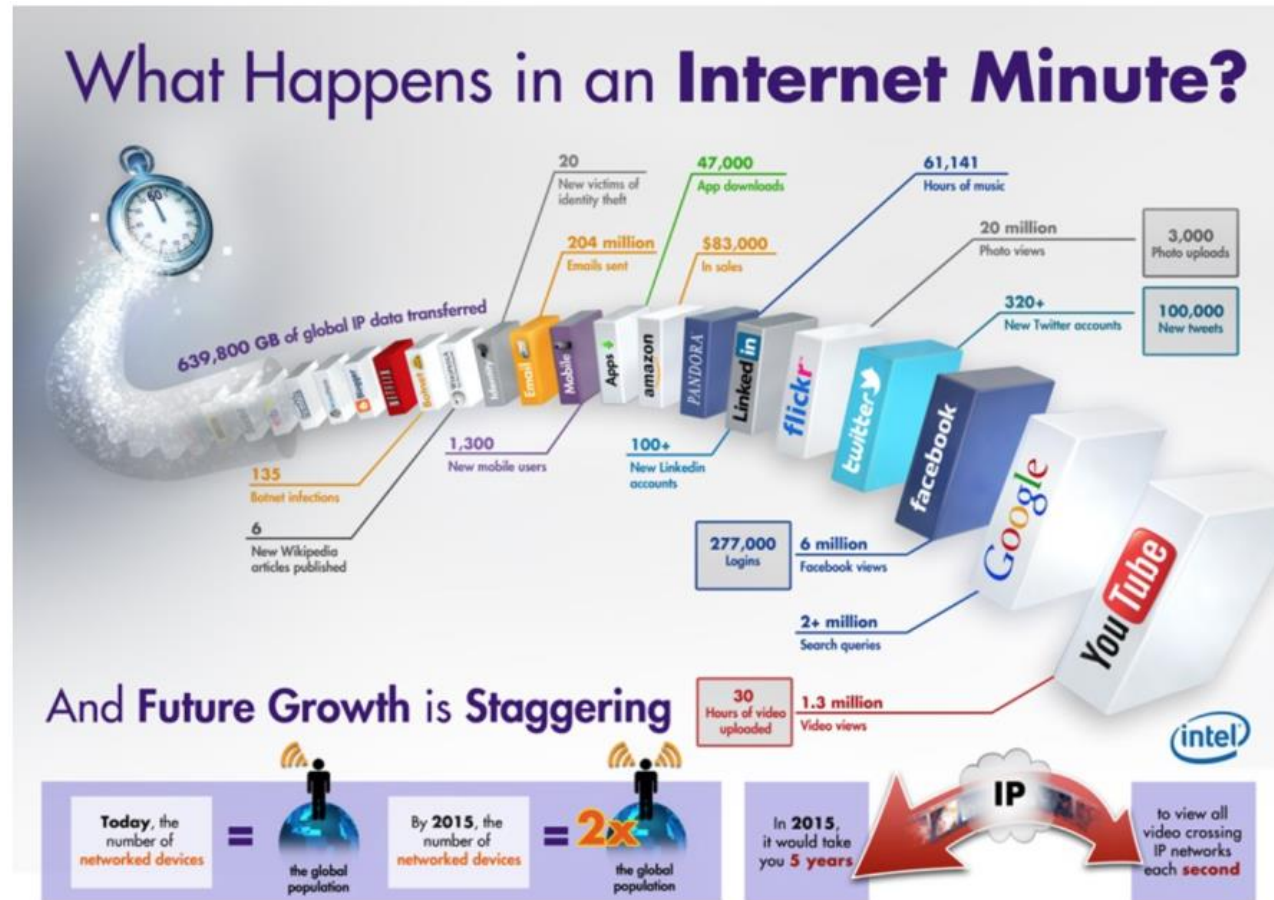


# Varie architetture in combinazione

Per rispondere alle molteplici esigenze in vari scenari d'uso sono necessari diversi tipi di piattaforme.



# Big Data, Cloud, IoT, Smart City

















Fonte: <https://junglemanager.wordpress.com>

Fonte: <http://clout-project.eu/>

# NoSQL

[www.nosql-database.org](http://www.nosql-database.org):

“Next Generation Databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and **horizontal scalable**. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: schema-free, easy replication support, simple API, eventually consistent/BASE (**not ACID**), a huge data amount, and more.”

Document Database	Graph Databases
  	 
Wide Column Stores	Key-Value Databases
   	    

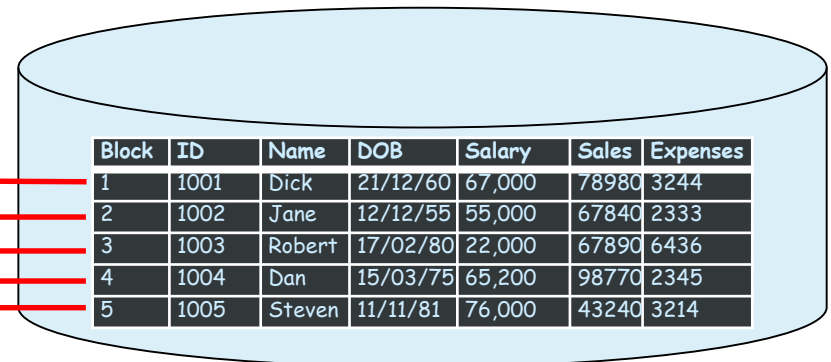
@cloudtxt <http://www.aryannava.com>

# Columnar architecture

Un **Column-oriented DBMS** è un sistema di gestione di basi di dati che memorizza i dati come sezioni di colonne di dati piuttosto che righe di dati. Ciò porta benefici nei sistemi Data Warehouse, nei sistemi CRM (Customer Relationship Management), nei cataloghi di schede bibliografiche e altri sistemi ad hoc, dove gli aggregati sono calcolati su un grande numero di dati simili tra loro.

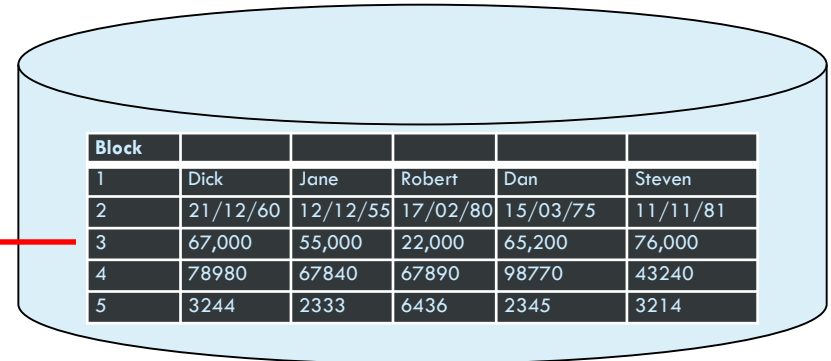
```
SELECT SUM(salary)
FROM SALE_PERSONS
```

**Row oriented  
database**



Block	ID	Name	DOB	Salary	Sales	Expenses
1	1001	Dick	21/12/60	67,000	78980	3244
2	1002	Jane	12/12/55	55,000	67840	2333
3	1003	Robert	17/02/80	22,000	67890	6436
4	1004	Dan	15/03/75	65,200	98770	2345
5	1005	Steven	11/11/81	76,000	43240	3214

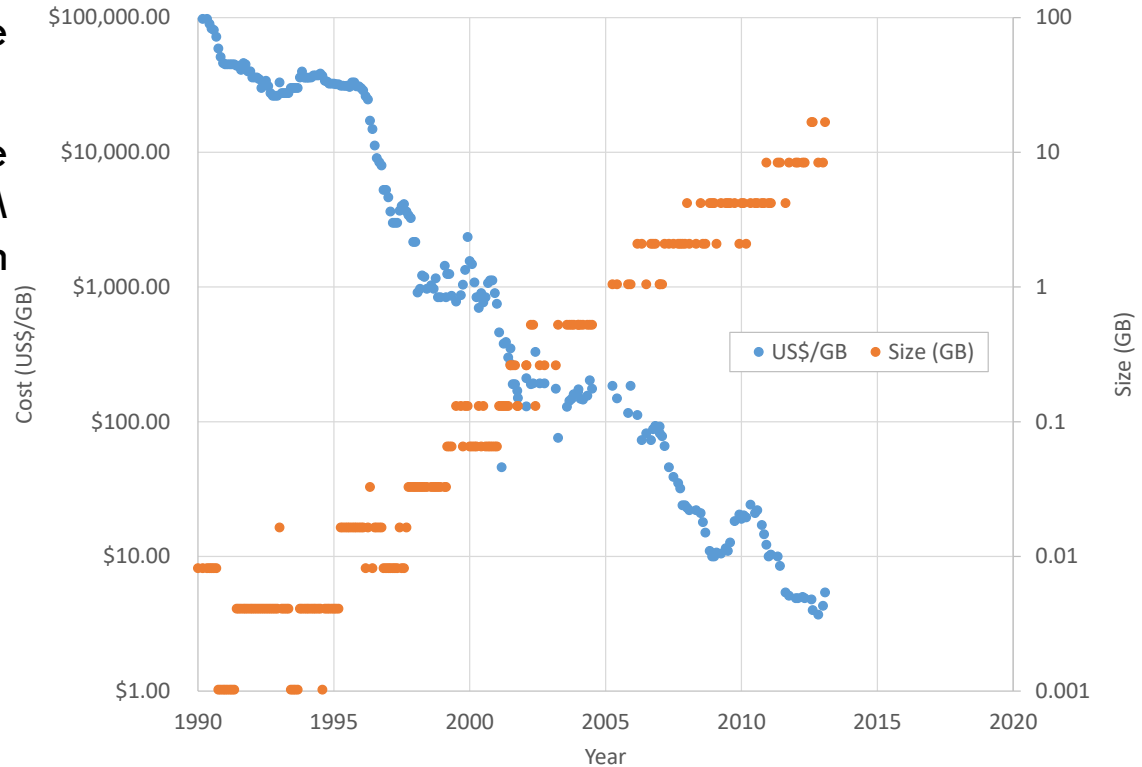
**Column oriented  
database**



Block						
1	Dick	Jane	Robert	Dan	Steven	
2	21/12/60	12/12/55	17/02/80	15/03/75	11/11/81	
3	67,000	55,000	22,000	65,200	76,000	
4	78980	67840	67890	98770	43240	
5	3244	2333	6436	2345	3214	

# The end of disk?

- Il costo della **RAM** diminuisce del 50% ogni 18 mesi.
- Alcuni DB possono essere ospitati interamente in RAM su singolo server o su un cluster di server.

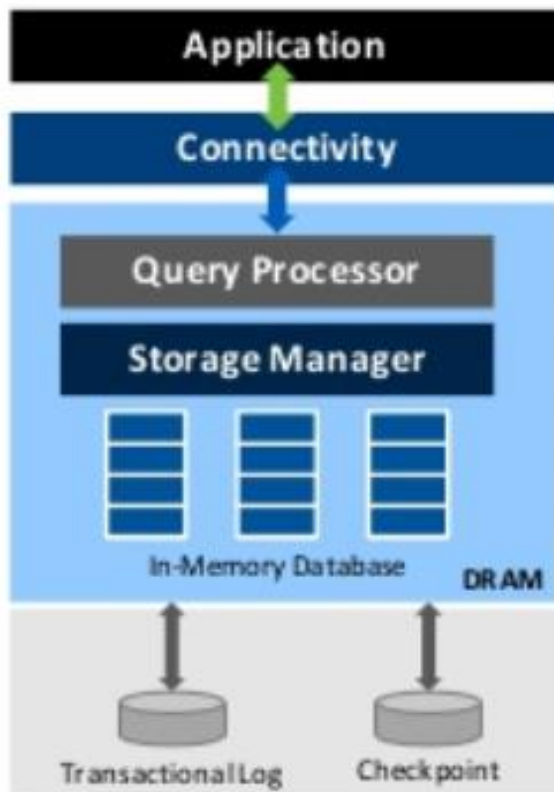


Inoltre i miglioramenti che si osservano nella tecnologia **Solid State Storage** fanno ben sperare per architetture che possano sfruttare un'appropriata gerarchia di memorie al fine di ottenere un soddisfacente rapporto costo/prestazioni per varie tipologie di applicazioni.

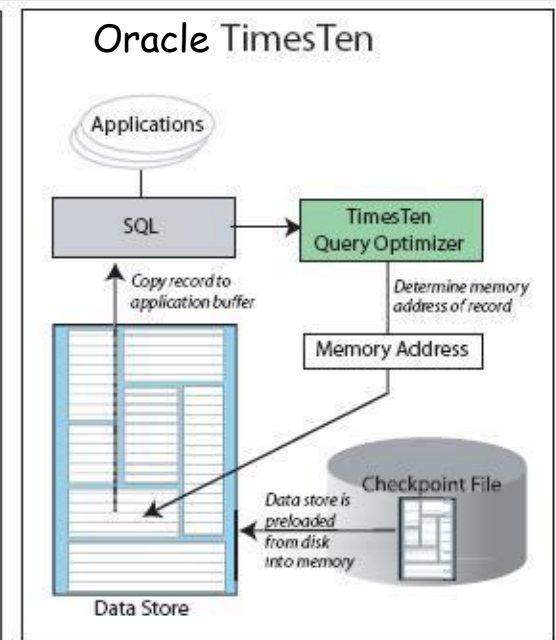
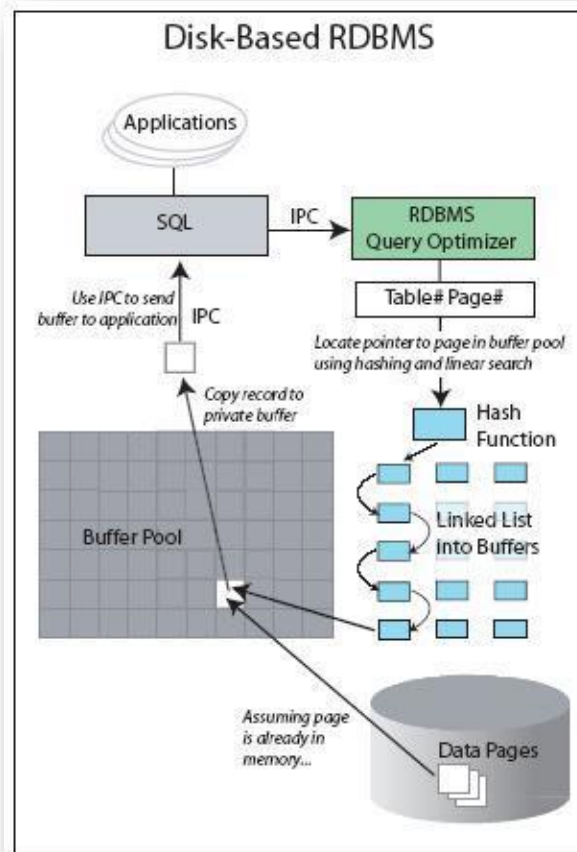


# In-Memory Data Base

**IMDBMS** (In-Memory Data Base Management System): sistema che gestisce basi di dati in memoria centrale; escludendo architetture onerose usualmente la dimensione del DB gestibile è molto inferiore rispetto a on-disk DB.



Fonte: Altibase

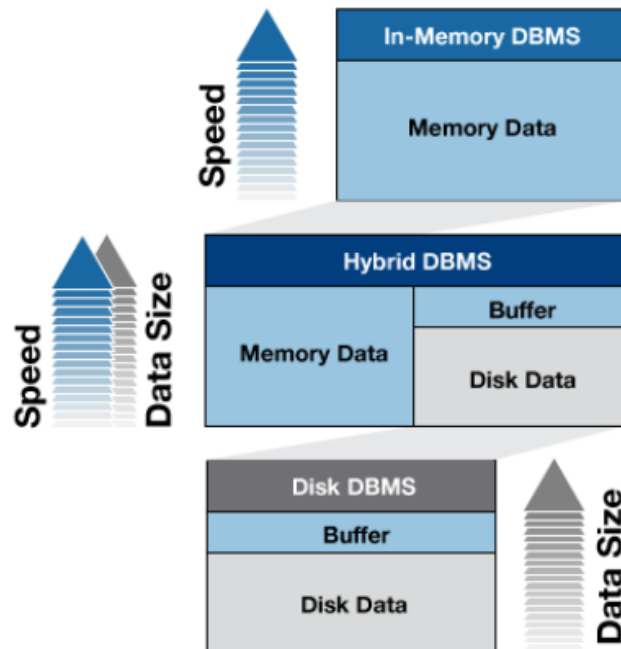


Fonte: Oracle



# Hybrid DBMS

**Hybrid DBMS:** un'architettura di DBMS più flessibile che consente di gestire una base di dati combinando entrambi i benefici offerti dalla memoria centrale, in termini di prestazioni, e dalla memoria secondaria in termini di capacità d'archiviazione, costo e persistenza.



Fonte: Altibase

Un esempio di architettura ibrida è Altibase HDB.

# Domande?

---

