Appunti di

Basi di Dati

SQL

HTML

PHP

Le basi di dati

Una base di dati (database) è un insieme di archivi di dati correlati tra loro e facilmente utilizzabili da un calcolatore

Le fasi di progettazione delle basi di dati sono:

- progettazione concettuale,
- progettazione logica
- progettazione fisica

Progettazione Concettuale

È il livello più alto della progettazione di un database. Durante la fase della progettazione concettuale della base di dati, il progettista ha il compito di estrapolare nei dettagli, con parole e linguaggi chiari e comuni, comprensibili a tutti, tutti i dati degli applicativi da sviluppare. Questa fase deve essere realizzata con strumenti indipendenti dall'ambiente e dal sistema di dati che si andrà ad utilizzare durante lo sviluppo.

Il metodo più semplice per riuscire in questa fase consiste nello scrivere una descrizione breve ed essenziale del problema e successivamente evidenziare tutti i **sostantivi** e i **verbi** presenti nel testo creato:

- i *sostantivi* saranno nella base di dati gli *oggetti*,
- *i verbi* rappresenteranno le *relazioni*.

Il prodotto dell'analisi è uno **schema concettuale**, schema che mette in relazione entità (gli oggetti) è il diagramma E/R (Entità / Relazioni) .

Questa rappresentazione deve essere indipendente da:

- i valori che verranno assegnati ai dati;
- le applicazione che utilizzeranno tali dati;
- le visioni parziali dei dati da parte degli utenti

Progettazione Logica

Durante la fase della progettazione logica si traduce lo schema concettuale prodotto nella fase di progettazione concettuale, nel modello di rappresentazione dei dati adottato dall'ambiente database scelto per la programmazione.

La struttura che daremo ai dati, in questa fase deve facilitare :

- La **manipolazione** dei dati (inserimenti e cancellazioni);
- l'**interrogazione** cioè la possibilità di ritrovare i dati, richiesti in modo semplice e veloce.

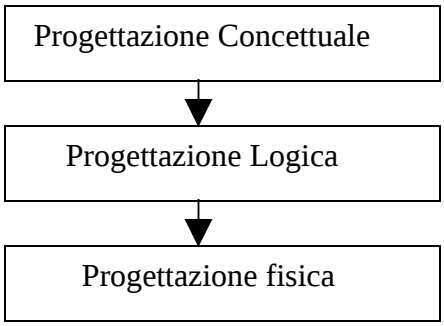
Questa fase descrive la composizione e tipo dei dati nel loro aspetto di struttura logica di dati. Il livello logico viene derivato da quello concettuale attraveso semplici regole. Il prodotto di questa fase è chiamato **schema logico**.

Progettazione Fisica

La terza ed ultima fase di progettazione di una base di dati è quella fisica, progettazione che in generale è affidata ai sistemisti e i DBA (Amministratori delle Basi di dati). Ogni forma di memorizzazione dipende soprattutto dall'ambiente che si è scelto di utilizzare e i software di gestione dei database si preoccupano di convertire lo schema logico prodotto nella fase di progettazione logica .

Metodologia

La metodologia di progettazione di una base di dati può vedersi quindi come un insieme di attività tra di loro collegate, finalizzate a produrre un insieme di prodotti intermedi e finali utilizzando criteri di verifica (normalizzazione) di qualità delle fasi e dei risultati ottenuti, volti a realizzare un database corretto e funzionale a partire da un insieme di specifiche legate al progetto di partenza.



Progettazione concettuale: il modello entità-associazioni

Il modello entità-associazioni è uno strumento per l'analisi delle caratteristiche di una applicazione indipendentemente dagli eventi che in essa accadono e dall'ambiente il cui si svilupperà poi la progettazione logica. Questo modello aiuta a ridurre la ridondanza dei dati e ad aumentare l'affidabilità e l'efficienza della applicazione. Il risultato di questa attività è una rappresentazione grafica, detta **schema E/R** (**Entity/Relationship**), che mette in evidenza gli aspetti fondamentali del modello concettuale con i dati e le relazioni fra essi.

Il modello entità associazioni fornisce un metodo per visualizzare le relazioni che intercorrono tra entità in una applicazione facilitando il passaggio dalla descrizione delle informazioni di una applicazione allo schema formale dei dati di un database.

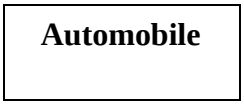
Gli elementi di un modello entità / associazioni sono

- entità
- associazioni
- attributi

Entità: un entità è una oggetto astratto o concreto che esiste nell'ambiente di cui si sta creando il modello ed è distinguibile dagli altri oggetti. Ad esempio: *amore, odio, auto.*

- **istanza di entità:** Una istanza di entità è una particolare occorrenza di una entità. Ad esempio Giuseppe Rossi è un' istanza dell'entità *studente*.
- **tipo o insieme di entità:** Un gruppo di entità simili costituisce un insieme di entità. Un insieme di entità ha proprietà comuni che sono condivise da tutte le istanze che la compongono. Ad esempio: il tipo di entità *astri* comprende *sole* e *luna*, il tipo di entità *sentimenti* comprende *amore* e *odio*.

L'entità viene rappresentata graficamente con un rettangolo:



Nel rettangolo è inserito un identificatore che descrive sinteticamente il significato dell' entità.

Attributo: Un attributo descrive una proprietà di un insieme di entità e di una associazione comune a tutte le istanze. Ad esempio: possibili attributi dell'insieme di entità *persone* sono *nome*, *cognome*, *data_di_nascita*. Le caratteristiche di un attributo sono:

- **formato** cioè carattere, numero o data/ora
- **dimensione** numero di caratteri o cifre se numerico
- **obbligatorietà** può assumere due valori *si* e *no* se si vuol dire che l'attributo deve sempre avere un valore, no altrimenti

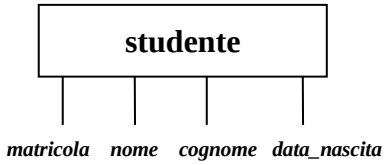
Gli attributi possono essere classificati in:

- **attributo semplice:** proprietà costituita da una singola unità di informazione indivisibile. Ad esempio: un possibile attributo semplice dell'insieme di entità *automobili* è *marca*.
- **attributo composto:** gruppo di attributi che possono essere considerati sia insieme che separatamente. Ad esempio: nell'insieme di entità *persone* l'attributo *indirizzo* è composto dagli attributi *via,numero,città,provincia,cap*
- **attributo multiplo:** due o più simili. Ad esempio: i voti di uno studente in una determinata disciplina.

Vedremo in seguito che sarà necessario modificare lo schema per eliminare attributi multipli e composti.

- **dominio di un attributo:** insieme dei valori che un attributo può assumere.

Graficamente gli attributi vengono rappresentati :



Un particolare attributo o insieme di attributi prende in nome di **chiave primaria** se identifica univocamente una particolare istanza di un insieme di entità . Esempi di chiave primaria sono il codice fiscale per l'entità cittadino, la matricola per l'entità studente. Le sue proprietà sono:

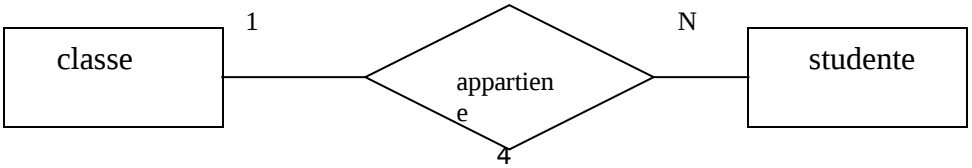
- *deve essere unica nel dominio;*
- *essere sempre presente;*
- *non cambiare nel tempo*.

Se non esiste un attributo o un insieme di attributi che soddisfano alle proprietà precedenti deve essere creata artificialmente (ad esempio il codice fiscale).
Una **chiave esterna** è un'attributo o un insieme di attributi che collega una istanza ad una istanza di un'altra entità attraverso la chiave primaria di quest'ultima. Le chiavi esterne devono rispettare le condizioni **integrità referenziale** *cioè se esiste una chiave esterna in un insieme di entità deve esistere una corrispondente chiave primaria nell'insieme di entità associato*.

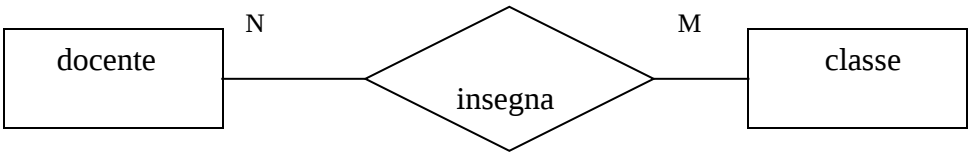
Associazione: Una associazione è una relazione tra insiemi di entità. Ad esempio: una associazione tra *persone* e *automobili* potrebbe essere *possiede* cioè l'indicazione di quale istanza dell' entità persone possiede una certa istanza dell' entità automobili. Le associazioni possono essere di tre tipi:

- **associazione uno-a-uno (1:1)**: per ogni istanza di entità in un insieme di entità c'è uno ed una sola istanza di entità associata in un altro insieme di entità. Per esempio: per ogni marito c'è la massimo una solo moglie e viceversa.

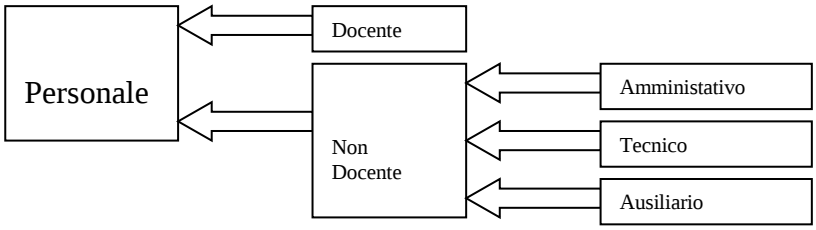
- **associazione uno-a-molti (1:N)**: ad ogni istanza di un' entita E1 è associata con *più istanze o nessuna* di un' entità E2 ma ad ogni istanza di entità E2 è associato al massimo con *una istanza* dell' entità E1. Per esempio: una donna può avere molti figli ma un figlio può avere solo una madre.



- **associazione molti-a-molti (N:M)**: ad ogni istanza di un' entita E1 è associata con *più istanze o nessuna* di un' entità E2 e viceversa. Un esempio di associazione molti-a-molti la associazione tra classi e professori. Ogni classe ha molti professori ma ogni professore ha molte classi.



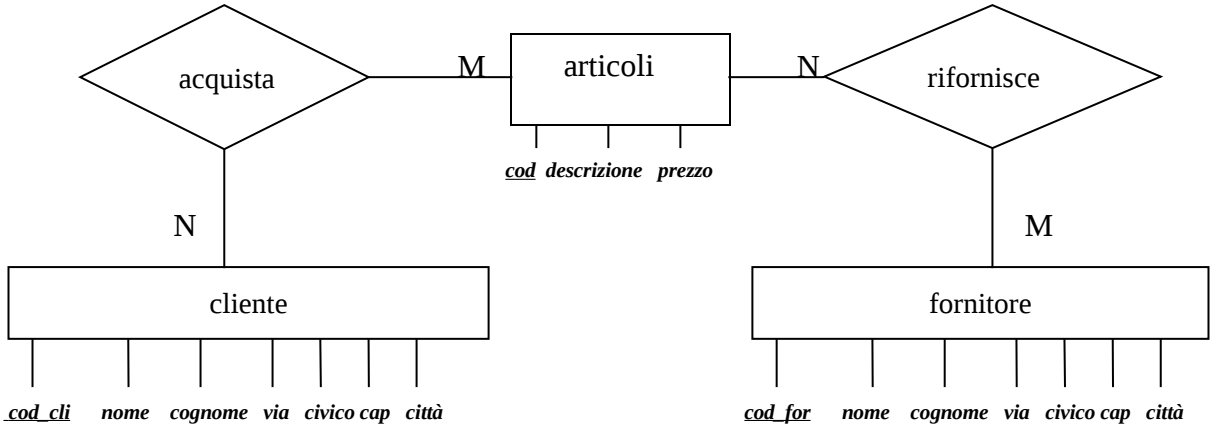
- **associazione gerarchica (IS-A):** E' una speciale associazione che consente ad un insieme che dipende gerarchicamente da un altro di ereditarne le proprietà.



Il personale di una scuola costituisce un insieme di entità che può essere suddiviso in due sottoinsiemi del personale docente e non-docente. Il personale non-docente può a sua volta essere suddiviso in amministrativo, tecnico e ausiliario. L'insieme di entità Personale ha proprietà comuni a tutti i sottoinsiemi (nome, cognome, indirizzo …) che tutti i sottoinsiemi ereditano. Inoltre ogni sottoinsieme ha proprietà specifiche non possedute dagli altri.

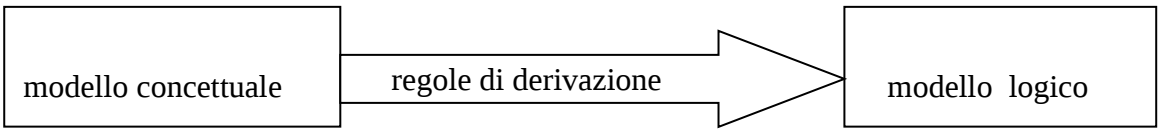
ESEMPIO

Creare un modello di un magazzino di una ditta di ricambi. Oltre ai dati degli ***articoli*** *in magazzino , si devono organizzare i dati relativi ai* ***clienti*** *che* ***acquistano*** *i ricambi ed ai* ***fornitori*** *che* ***riforniscono*** *il magazzino di ricambi*



Progettazione logica : le regole di derivazione del modello logico

Dal modello concettuale è possibile ottenere il modello logico dei dati: più semplicemente si può definire la struttura degli archivi adatti per organizzare i dati.



Come già detto tali strutture devono facilitare la manipolazione e l'interrogazione dei dati. Il modello logico deriva da quello concettuale applicando le seguenti regole:

1. ogni entità diventa un' archivio;
2. ogni attributo di entità diventa un campo dell'archivio;
3. ogni campo eredita tipo, lunghezza e obbligatorietà dell'attributo da cui deriva;
4. la chiave primaria dell'entità diventa chiave primaria del record dell'archivio derivato;
5. l'associazione 1:1 diventa un unico archivio i cui campi sono gli attributi delle due entità;
6. l'associazione 1:N si realizza con due archivi ciascuno con i campi relativi agli attributi dell'entità da cui derivano. In particolare nell' archivio della parte N vi sarà un nuovo campo: la chiave primaria dell' archivio della parte a 1 (detta **chiave esterna**);
7. l'associazione N: M si realizza con tre archivi i primi due derivano dalle due entita, il terzo è un nuovo archivio che ha come chiavi primarie le chiavi primarie dei due archivi che mette in relazione e gli eventuali attributi della relazione.

ESEMPIO

articoli	**cod**	**descrizione**	**prezzo**
tipo	numerico	alfanumerico	numerico con 2 decimali
lunghezza	5	40	7.2
obbligatorietà	si	si	no

cliente	**codice**	**nome**	**cognome**	**indirizzo**
tipo	numerico	alfanumerico	alfanumerico	alfanumerico
lunghezza	10	30	30	50
obbligatorietà	si	si	si	si

fornitore	**codice**	**nome**	**cognome**	**indirizzo**
tipo	numerico	alfanumerico	alfanumerico	alfanumerico
lunghezza	10	30	30	50
obbligatorietà	si	si	si	si

Le tre entità articoli, cliente e fornitore diventano archivi e gli attributi diventano campi e la chiave primaria resta tale anche per gli archivi.
Le due associazioni, essendo di tipo N:M diventano anch'esse archivi

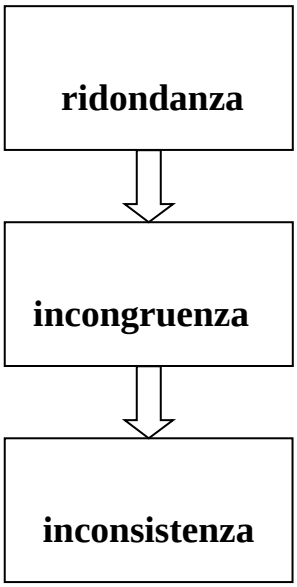
acquista	**cod**	**codice (del cliente)**
tipo	numerico	numerico
lunghezza	5	10
obbligatorietà	si	si

rifornisce	**cod**	**codice (del fornitore)**
tipo	numerico	numerico
lunghezza	5	10
obbligatorietà	si	si

Con il *termine **base di dati** si indicano archivi correlati di dati, organizzati in modo integrato attraverso tecniche di modellazione dei dati e gestiti sulle memorie di massa attraverso appositi software con l'obiettivo di raggiungere una grande efficienza in termini di velocità, di consistenza, di sicurezza e di integrità delle registrazioni*.

- **velocità** di reperimento dei dati richiesti;
- **consistenza**: cioè i dati in essi contentuti devono essere significativi ed essere effettivamente utilizzabili nelle applicazioni dell'azienda;
- **integrità** : si deve garantire che le operazioni effettuate sul database da utenti autorizzati non provochino una perdita di consistenza ai dati;
- **sicurezza** : si deve impedire che il database venga danneggiato da interventi accidentali o non autorizzati.

I software per la gestione di database vengono indicati con il termine **Data Base Management System (DBMS**) o sistema per la gestione delle basi dati.
I concetti di integrazione degli archivi e di base di dati forniscono l'idea di dati accentrati. In realtà gli archivi integrati che costituiscono la base di dati aziendali possono risiedere su un unico computer oppure possono essere distribuiti sulle memorie di massa di computer diversi facenti parte di una rete aziendale, i cui nodi possono essere anche fisicamente lontani: in questo caso si parla di *database distribuiti*.
Al fine di superare i limiti dei sistemi di archiviazione non integrati dei primi anni '60, nei quali spesso si avevano inconvenienti di



ridondanza dati che compaiono in maniera duplicata su più archivi e che può portare all'incongruenza
incongruenza *valori diversi* dello *stesso dato* in archivi diversi che porta all' inconsistenza
incostistenza inutilizzabilità dei dati perché non affidabili

A partire dagli anni '70 la teoria dei database introduce una nuova metodologia. Le caratteristiche principali di questa gestione sono:
1) Indipendenza dalla struttura fisica dei dati
Si possono essere modificare supporti e metodi di accesso ai dati senza modificare il software applicativo.
2) Indipendenza dalla struttura logica dei dati

E’ possibile apportare modifiche alle strutture dei dati senza modificare il software applicativo.

3) Utilizzo da parte di più utenti

Utenti diversi possono accedere ai dati in modo concorrente senza interferire fra loro e garantisce l’accesso specifico ad alcune tabelle a seconda dell’utenza.

4) Eliminazione della ridondanza

Gli stessi dati non compaiono più volte in archivi diversi.

5) Eliminazione della inconsistenza

Il database non può presentare campi uguali con valori diversi in archivi diversi

6) Facilità di accesso

Il ritrovamento dei dati è veloce e semplice.

7) Integrità dei dati

Vengono previsti controlli per evitare anomalie ai dati causate dalle applicazioni degli utenti;

8) Sicurezza dei dati

Sono previste precedure di controllo per impedire accessi non autorizzati ad dati contenuti nel database e i protezione da guasti accidentali.

9) Uso di lunguaggi per la gestione del database

Il database viene gestito attraverso:

- comandi di manipolazione (inserimento, modifica e cancellazione) dei dati contenuti in esso
- comandi per le interrogazioni per ottenere le informazioni desiderate.

I comandi non agiscono su un singolo record per volta, ma su gruppi di record per volta.

Modelli di database

A partire dallo schema concettuale E/R, un database può essere progettato e realizzato passando al modello logico, cioè alle strutture di dati che consentono di organizzare i dati per consentire le operazioni di manipolazione e di interrogazione.

La soluzione più semplice consiste nel costruire un database con una struttura di dati formata da un Nello sviluppo della teoria dei database, dal 1960 in poi, sono emersi principalmente tre tipi diversi di modelli per le basi di dati:

Modello Gerarchico

E’ particolarmente adatto per rappresentare situazioni nelle quali è possibile fornire all’insieme dei dati una struttura nella quale ci sono entità che stanno in alto ed entità che stanno in basso, secondo uno schema ad albero, nel quale i nodi rappresentano le entità e gli archi rappresentano le associazioni. Nella pratica l’entità è un file, l’istanza è un record e gli attributi sono i campi del record. E’ particolarmente adatto a rappresentare le associazioni 1:N. Presenta dei limiti soprattutto nella rigidità della struttura di dati creata, che talvolta non riesce ad evitare ridondanze dei dati.

Modello Reticolare

In questo modello le entità rappresentano i nodi e le associazioni rappresentano gli archi di uno schema a grafo orientato: si tratta cioè di una estensione del modello gerarchico. Non esiste quindi una gerarchia predefinita tra le entità, un record figlio può avere un numero qualsiasi di padri: in questo modo vengono evitate situazioni di ripetizione di dati uguali.

Risulta però più difficile l’implementazione e la costruzione del software applicativo. Ha avuto una discreta diffusione negli anni ’70 soprattutto nei sistemi di grandi dimensioni.

Modello Relazionale

Rappresenta il database come un insieme di tabelle. Esso viene considerato attualmente il modello più semplice ed efficace, perché è più vicino al modo consueto di pensare i dati, e si adatta in modo naturale alla classificazione e alla strutturazione dei dati.

Il modello relazionale nasce nel 1970, proposto da Edward F. Codd, ricercatore dell'IBM come idea di un modello logico molto semplice e nello stesso tempo in grado di superare i limiti degli altri modelli utilizzati. Entra a far parte di sistemi commerciali a partire dal 1978. Esso si basa su alcuni concetti fondamentali della matematica e assegna grande importanza all'uso rigoroso del linguaggio matematico, con due obiettivi importanti:

- utilizzare un linguaggio conosciuto a livello universale, qual'è il linguaggio matematico
- eliminare i problemi di ambiguità nella terminologia e nella simbologia

Con l'introduzione di questo modello i modelli gerarchico e reticolare sono presto diventati obsoleti per il mercato e per la ricerca. Questo perché le operazioni sui database gerarchici sono complesse, agiscono sui singoli record e non su gruppi di record. I modelli non relazionali sono fondamentalmente basati sulla programmazione di applicazioni, l'utente deve specificare i percorsi per ritrovare i dati e la velocità nel ritrovare le informazioni dipende dal software di gestione; nel modello relazionale invece i percorsi per le interrogazioni sono a carico del sistema.
Il modello relazionale è più intuitivo e più espressivo per la strutturazione dei dati, rispetto agli altri modelli però è relativamente più lento, nella ricerca e occupa più spazio su memoria di massa rispetto ai database creati e gestiti da DBMS basati sugli altri modelli.

Il Modello Relazionale nel dettaglio.
Si chiama così perché è fondato sul concetto matematico di relazione tra insiemi di oggetti.

Dati n insiemi $A1, A2,\ldots\ldots, An$, si dice **relazione** e si indica con **R ($A1, A2\ldots\ldots, An$) ,** un sottoinsieme dell'insieme di tutte le n-uple $(a_1,\ldots\ldots,a_n)$ che si possono costruire prendendo nell'ordine un elemento $a_1 \quad A1, \quad a_2 \quad A2, \ldots\ldots\ldots \quad a_n \quad An.$
Il numero n si chiama **grado** della relazione, gli insiemi Ai si chiamano **domini** della relazione, e il numero delle n-uple o **tuple** si chiamano **cardinalità** della relazione.

La relazione viene rappresentata con una tabella, avente

- tante colonne quanti sono i domini (grado della relazione) ;
- tante righe quante sono le n-uple (cardinalità della relazione);
- i nomi dei domini sono i nomi delle colonne;
- i valori che compaiono in una colonna sono omogenei tra loro, cioè appartengono a uno stesso dominio.

La relazione è quindi una collezione di n-uple, ciascuna delle quali contiene i valori di un numero prefissato di colonne. La relazione rappresenta un'entità, ogni n-upla rappresenta un'istanza dell'entità, le colonne contengono gli attributi dell'entità, il dominio è l'insieme dei valori che possono essere assunti da un attributo.
La **chiave della relazione** *è un insieme di uno o più attributi che identificano univocamente le n-uple all'interno della relazione*, cioè ogni riga della tabella possiede valori diversi per l'attributo (o gli attributi) chiave.

Un esempio di relazione è STUDENTE *(<u>matricola</u>, cognome, nome, indirizzo, telefono)*
Rappresentata in forma di tabella come:

<u>matricola</u>	***cognome***	***nome***	***indirizzo***	***telefono***
19238	Rossi	Giuseppe	Via Roma, 9	02345678
35895	Verdi	Mario	Piazza Verdi, 1	01456789
12993	Bianchi	Nicola	Casella postale 99	099456723

Requisiti del Modello Relazionale
In sistesi sono:

1. tutte le righe della tabella contengono lo stesso numero di colonne, corrispondente agli attributi;
2. gli attributi rappresentano informazioni elementari , non scomponibili ulteriormente;
3. i valori assunti da un campo appartengono al dominio dei valori possibili per quel campo, e quindi sono valori omogenei tra loro, cioè sono dello stesso tipo;
4. in una relazione, ogni riga è diversa da tutte le altre, cioè non ci possono essere due righe con gli stessi valori dei campi: questo significa che esiste un attributo o una combinazione di più attributi che identificano univocamente la n-upla, e che assumono perciò la funzione di chiave primaria della relazione;
5. non è importante l'ordine delle righe nella tabella.

Il modello relazionale fissa una regola di integrità sui dati, detta **integrità sull'entità** (entity integrity) secondo la quale:

nessun attributo della chiave primaria può avere valore nullo

Le regole di derivazione applicabili al modello E/R sono le stesse che abbiamo visto sopra per il modello logico in generale e cioè.

1. ogni entità diventa una relazione;
2. ogni attributo di entità diventa un attributo della relazione;
3. ogni attributo eredita tipo, lunghezza e obbligatorietà dell'attributo da cui deriva;
4. La chiave primaria dell'entità diventa chiave primaria della tupla della relazione derivata;
5. l' associazione 1:1 diventa un unica relazione i cui attributi sono gli attributi delle due entità;
6. l'associazione 1:N si realizza con due relazioni ciascuno con gli attributi relativi agli attributi dell'entità da cui derivano. In particolare nella relazione della "parte a N" vi sarà un nuovo campo: la chiave esterna della relazione della "parte a 1";
7. l'associazione N:M si realizza con tre relazioni le prime due derivano dalle due entita, la terza è un nuovo archivio che ha come chiavi primarie le chiavi primarie delle due relazioni che associa e gli eventuali attributi della relazione.

Le operazioni relazionali

*In generale gli **operatori relazionali** agiscono su una relazione per ottenere una nuova relazione*. In sostanza le operazioni relazionali consentono di effettuare le interrogazioni alla base di dati per ottenere le informazioni desiderate estraendo da una tabella una sottotabella, oppure combinando tra loro due o più tabelle e generando così nuove relazioni.

Due relazioni R e S si dicono **congruenti** quando sono dello stesso stesso grado e gli attributi di stessa posizione sono dello stesso tipo.

Le principali operazioni sono:

L'unione fra due relazioni R e S congruenti $\mathbf{R \cup S}$ produce una nuova relazione costituita dall' insieme delle righe di R e di S.

Grado: lo stesso delle relazioni di partenza

Cardinalità: cardinalità di R sommata a quella di S

L'intersezione fra due relazioni R e S congruenti $\mathbf{R \cap S}$ produce una nuova relazione costituita dalle righe comuni sia a R che a S.

Grado: lo stesso delle relazioni di partenza

Cardinalità: sconosciuta

La differenza due relazioni R e S congruenti **R – S** produce una nuova relazione costituita dalle righe di R che non sono anche righe di S.
Grado: lo stesso delle relazioni di partenza
Cardinalità: sconosciuta
Esempio:

A	*B*	*C*
b	g	a
d	a	f
c	b	d

Relazione S

Relazione R

A	*B*	*C*
a	b	c
d	a	f
c	b	d
c	b	f
b	g	a
d	a	f
c	b	d

Relazione R ∪ S

A	*B*	*C*
d	a	f
c	b	d

Relazione R ∩S

A	*B*	*C*
a	b	c
c	b	f

Relazione R – S

La proiezione su una relazione R $\pi_{a, b}$ **(R)** genera una nuova relazione estraendo dalla tabella iniziale due o più colonne corrispondenti agli attributi prefissati.
Grado: minore o uguale al grado della relazione di partenza
Cardinalità: di solito uguale a quella di partenza tranne quando vengono ridotte a una le righe uguale con la unique.
Esempio

A	*B*	*C*	*D*	*E*
a	b	c	b	g
d	a	f	b	g
c	b	f	b	g
a	b	c	d	a
d	a	f	d	a
c	b	f	d	a

Relazione R

A	*B*	*C*
a	b	c
d	a	f
c	b	f

Relazione $\pi_{A,B,C}$ (R)

B	*D*
b	b
a	b
b	d
a	d

Relazione $\pi_{B,D}$ (R)

La selezione su una relazione R $\sigma_{a = valore}$**(R)** genera una nuova relazione è costituita solo dalle n-uple della relazione di partenza che soddisfano a una determinata condizione.
Grado: lo stesso della relazione di partenza
Cardinalità: minore o uguale a quella di partenza
Esempio

A	*B*	*C*	*D*	*E*
a	b	c	b	g
d	a	f	b	g
c	b	f	b	g
a	b	c	d	a
d	a	f	d	a
c	b	f	d	a

Relazione R

A	*B*	*C*	*D*	*E*
a	b	c	b	g
a	b	c	d	a

Relazione $\sigma_{A = a}(R)$

A	*B*	*C*	*D*	*E*
a	b	c	b	g

Relazione $\sigma_{A = a \text{ and } D = b} (R)$

Prodotto cartesiano fra due relazioni R e S **R X S** produce una nuova tabella costituita dalle righe di R concatenate alle righe di S.
Esempio

A	*B*	*C*
a	b	c
d	a	f
c	b	f
a	c	d

Relazione R

D	*E*
b	d
d	a
b	a

Relazione S

A	*B*	*C*	*D*	*E*
a	b	c	b	d
a	b	c	d	a
a	b	c	b	a
d	a	f	b	d
d	a	f	d	a
d	a	f	b	a
c	b	f	b	d
c	b	f	d	a
c	b	f	b	a
a	c	d	b	d
a	c	d	d	a
a	c	d	b	a

Relazione **R X S**

La congiunzione fra due relazioni R e S aventi l'attributo **a** in comune **R |X|** $_{\text{R.a = S.a}}$ **S** genera una nuova relazione olttenuta :

- facendo il prodotto cartesiano **R X S**
- alla relazione ottenuta al punto precedente $\sigma_{\text{R.a = S.a}}$ **(R x S)**

in sintesi si genera una nuova relazione che contiene le righe della prima e della seconda tabella, che possono essere combinate secondo i valori uguali dell'attibuto comune.
Grado: se i gradi delle relazioni di partenza sono rispettivamente N1 e N2, il grado della relazione generata è uguale a N1+N2-1, perché l'attributo comune compare una sola volta.
Cardinalità: non è prevedibile in quanto si ottengono solo le righe che possono essere combinate attraverso valori presenti in entrambe.

Esempio: Partendo dal risultato del precedente esempio e applicando la sola selezione alla relazione **R X S** per definizione di join avremo:

A	*B*	*C*	*D*	*E*
a	b	c	b	d

A	*B*	*C*	*D*	*E*
a	b	c	b	d

c	b	f	b	d
a	b	c	b	a
c	b	f	b	a

Relazione **R |X|** $_{B=D}$ **S**

Relazione **R |X|** $_{C<E}$ **S**

Esercizi

Facendo riferimento all'esempio del magazzino ricambi, svolgere i seguenti esercizi:

1. L'elenco di tutti i clienti di cognome 'Rossi'
2. L'elenco di tutti i fornitori di Roma
3. Nome e cognome di tutti i clienti di Firenze e Bologna
4. Tutti gli articoli il cui prezzo è maggiore di 120 euro
5. L'elenco di tutti gli articoli acquistati dal cliente di codice 340
6. Codice e descrizione di tutti gli articoli forniti dal fornitore di codice 24
7. Codice e descrizione di tutti gli articoli acquistati dal cliente di cognome 'Mazzini'
8. L'elenco di tutti gli articoli acquistati dal cliente di codice 44. Per ciascun articolo dovrà esser specificato il nome e cognome di un fornitore.

Il processo di Normalizzazione delle relazioni

Nella definizione della struttura di una relazione occorre evitare la ripetizione e la ridondanza dei dati per non creare problemi nella fase di trattamento della tabella con operazioni di modifica o cancellazione di righe.

La normalizzazione è un processo con il quale le tabelle vengono trasformate in altre tabelle in modo tale che a ciascuna di esse corrisponda un singolo oggetto della realtà rappresentata con il modello di database. Le regole della normalizzazione sono definite per evitare l'inconsistenza dei dati e le anomalie nelle operazioni di aggiornamento, inserimento e cancellazione.

Ricordiamo le definizioni di:

- la **chiave o chiave primaria** è l'insieme di uno o più attributi che identificano in modo univoco una n-upla
- **l'attributo non-chiave** è un campo che non fa parte della chiave primaria;

Introduciamo le definizioni seguenti

- **la dipendenza funzionale** tra attributi si ha quando il valore di un attributo A1 determina un singolo valore dell'attributo A2 e si indica con A1 → A2
- **la dipendenza transitiva** si ha quando un attributo A2 dipende da A1 e l'attributo A3 dipende da A2; allora A3 dipende transitivamente da A1

se A1 → A2 e A2 → A3 allora A1 → A3

Prima Forma Normale

Una relazione è in prima forma normale (**1FN**) quando rispetta i requisiti fondamentali del modello relazionale che sono:

1. gli attributi rappresentano informazioni elementari cioè *non devono* essere multivalore o strutturati (cioè costituire essi stessi una tupla);
2. tutte le righe della tabella devono contenere lo stesso numero di colonne;
3. i valori che compaiono in una colonna sono dello stesso tipo;
4. ogni riga è diversa da tutte le altre;
5. l'ordine con il quale le righe compaiono nella tabella è irrilevante.

Consideriamo ad esempio la relazione *studente(matricola, nome, classe, materia, interrogazioni)*

matricola	***nome***	**classe**	***materia***	***interrogazioni***
1234	Rossi Mario	V A	matematica	10/12/2010 5
7890	Verdi Luigi	IV B	italiano	10/12/2010 5
1234	Rossi Mario	V A	italiano	11/11/2010 8 03/07/2010 7

L'attributio *classe* non è elementare infatti è costituito da anno e sezione;
L'attributio *nome* non è elementare infatti è costituito da nome e cognome;
L'attributo *interrogazione* è strutturato perché per ogni interrogazione sono riportate due informazioni data e voto;
L'attributo *interrogazione* è multivalore perché potremo avere più interrogazioni nella stessa disciplina in date differenti;
La relazione studente non è quindi in 1NF per strutturarlo dovremo trasformare ogni attibuto strutturato o multivalore in più attributi semplici. Nel caso dell'attributo multivalore dovremo aggiungere tante tuple con gli attributi multivalore divenuti semplici. Quindi la relazione studente diviene: *studente(matricola, nome,cognome, sezione, classe, materia, data, voto)*

matricola	***nome***	***cognome***	***sezione***	***classe***	***materia***	***data***	***voto***
1234	Mario	Rossi	A	V	matematica	10/12/2010	5
7890	Luigi	Verdi	B	IV	italiano	10/12/2010	5
1234	Mario	Rossi	A	V	italiano	11/11/2010	8
1234	Mario	Rossi	A	V	italiano	03/07/2010	7

Le forme normali superiori alla prima vengono introdotte per eliminare problemi durante le operazioni di aggiornamento o di cancellazione, evitando l'inconsistenza o la perdita indesiderata di dati.

Seconda Forma Normale

Una relazione è in seconda forma normale (**2FN**) quando:
1. è in **1NF**;
2. tutti i suoi attributi non-chiave dipendono dall'intera chiave, cioè non ci sono attributi che dipendono soltanto da una parte della chiave.

La seconda forma normale elimina la dipendenza parziale degli attributi dalla chiave e riguarda il caso di relazioni con chiavi composte, cioè formate da più attributi.
Utilizzando la simbologia descitta nelle definizioni precedenti di dipendenza, il passaggio alla seconda forma normale si schematizza nel seguente modo:

Esempio: Una azienda di trasporti gestisce le corse di alcune linee identificate dai due capolinea mediante un certo numero di autobus (ogni autobus indentificato da un numero è catatterizzato da un certo numero di posti). Una possibile relazione *CORSE* è:

Ora	***Linea***	***Origine***	***Destinazione***	***Bus***	***Posti***
7.00	15	V. Verdi	Mazzini	30044	40
7.00	18	C.Battisti	P.Garibaldi	25250	20
7.30	15	V. Verdi	Mazzini	19154	70
8.00	18	C.Battisti	P.Garibaldi	30044	40
8.00	11	deposito	Stazione	25328	20
8.30	15	V. Verdi	Mazzini	25250	30

La chiave primaria è determinata dalla combinazione dell'ora di partenza e della linea.
La relazione non è in seconda forma normale perchè gli attributi Origine e Destinazione dipendono solo parzialmente dalla chiave (dipendono solo da Linea e non da Ora). La consegnuenza di questa

anomalia è che i dati di origine e destinazione sono ripetuti più volte nella stessa tabella. Il problema non è solo di ridondanza (eccesso di dati) ma è anche di consistenza (validità dei dati). Infatti se si cambia un capolinea la modifica va fatta in tutte le righe in cui compare altrimenti il capolinea può cambiare in funzione della riga visitata.
Per passare in seconda forma normale devono essere estratti dalla relazione tutti gli attributi che dipendono parzialmente dalla chiave e spostati in una nuova relazione in cui dipendono totalmente dalla chiave.

Ora	***Linea***	***Bus***	***Posti***
7.00	15	30044	40
7.00	19	25250	20
7.30	15	19154	70
8.00	19	30044	40
8.00	21	25328	20
8.30	15	25250	30

Linea	***Origine***	***Destinazione***
15	V. Verdi	Mazzini
18	C.Battisti	P.Garibaldi
11	deposito	Stazione

Terza Forma Normale

Una relazione è in terza forma normale (**3FN**) quando
1. è in **2NF**;
2. tutti gli attributi non-chiave dipendono direttamente dalla chiave, cioè non possiede attributi non-chiave che dipendono da altri attributi non-chiave.

La terza forma normale elimina la dipendenza transitiva degli attributi dalla chiave.
Nell' esempio precedente si vede che la prima delle due relazioni ottenute nel passaggio in seconda forma normale non è in terza forma normale. Infatti l'attributo posti non dipende direttamente dalla chiave (Ora+Linea) ma solo indirettamente attraverso l'attributo non-chiave Bus. L'anomalia che si determina è la ripetizione del valore di Posti ad ongi ripetizione di un valore di Bus. La trasformazione in terza forma normale porta a questa situazione:

Ora	***Linea***	***Bus***
7.00	15	30044
7.00	18	25250
7.30	15	19154

Linea	***Origine***	***Destinazione***
15	V. Verdi	Mazzini
18	C.Battisti	P.Garibaldi
11	deposito	Stazione

Bus	***Posti***
30044	40
25250	30
19154	70

Si deve osservare che il processo di normalizzazione diminuisce la ridondanza dei dati e la possibilità di inconsistenza, ma rende più complesse le operazioni di ritrovamento dei dati perché le query devono abbracciare più tabelle.
Il modello relazionale richiede come obbligatoria solo la prima forma normale. Tuttavia il passaggio alle forme normali superiori consente di distinguere e separare con precisione gli oggetti, senza perdita di informazioni, anche se viene generata una ridondanza di dati che però è costantemente sotto controllo.

Integrità Referenziale

Oltre alla Entity integrity che ci dice che nessun attributo della chiave può esser nullo abbiamo una seconda regola di integrità.

L’integrità referenziale (referential integrity) è un insieme di regole del modello relazionale che garantiscono l’integrità dei dati *quando si hanno relazioni associate tra loro attraverso la chiave esterna*: queste regole servono per rendere valide le associazioni tra le tabelle e per eliminare gli errori di inserimento, cancellazione o modifica di dati collegati tra loro.
L’integrità referenziale viene rispettata quando per ogni valore non nullo della chiave esterna, esiste un valore corrispondente della chiave primaria nella tabella associata

Quando viene applicata l’integrità referenziale, è necessario osservare le seguenti regole pratiche:

1. Non è possibile immettere un valore nella chiave esterna della tabella associata, se tale valore non esiste tra le chiavi della tabella primaria.

2. Non è possibile eliminare una n-upla dalla tabella primaria, se esistono righe legate ad essa attraverso la chiave esterna nella tabella correlata

3. Non si può modificare il valore della chiave nella tabella primaria, se ad essa corrispondono righe nella tabella correlata.

La gestione del database

Il **DBMS (DataBase Management System)** è il software che consente di costruire e gestire una base di dati, realizzandola nella pratica su memoria di massa, a partire da un progetto e da uno schema dei dati definiti a livello concettuale e tradotto poi in un modello logico dei dati.
Le funzioni che il DBMS è in grado di offrire sono:

1) Implementazione del modello logico sul sistema di elaborazione
2) Manipolazione e interrogazione sulla base di dati
3) Controllo dell’integrità dei dati
4) Sicurezza e protezione

Linguaggi per il database

I comandi che il DBMS mette a disposizione possono essere classificati nelle seguenti categorie di linguaggi:

1. **DDL (Data Definition Language)** linguaggio per la descrizione dei dati, delle tabelle e delle viste;
2. **DMCL (Device Media Control Language)** il linguaggio di controllo dei supporti di memorizzazione dei dati;
3. **DML (Data Manipulation Language)**, linguaggio per il trattamento (o manipolazione) dei dati contenuti nel database, che consente le operazioni di accesso per inserimenti, modifiche o cancellazioni di n-ple;
4. **DCL (Data Control Language)** linguaggio per fissare i vincoli di integrità, per stabilire le autorizzazioni agli accessi e i tipi di permessi consentiti agli utenti (inserimento di nuovi dati, solo lettura, modifica dei dati),
5. **Query Language** linguaggio per le interrogazioni alla base di dati, che consente il ritrovamento dei dati che interessano, sulla base dei criteri di ricerca richiesti dall’utente.

I comandi del linguaggio relazionale operano inoltre su gruppi di righe o sull'intera tabella, anziché su una riga per volta: con una sola richiesta possono essere trattati o ritrovati molti record e non solo un record per volta, come avviene con i tradizionali linguaggi di programmazione.

Gli Utenti

Un database viene utilizzato da persone diverse, per funzioni e per applicazioni diverse:

1. la responsabilità della gestione del DB è affidata **all'Amministratore della Base di Dati** (**DBA, Database Administrator**), con i seguenti compiti: implementazione del modello logico del db nel sistema di elaborazione sui supporti fisici delle memorie di massa
2. i **Programmatori** che intendono utilizzare per le loro applicazioni i dati organizzati in un db, utilizzano un linguaggio DML, oppure comandi che sono un'estensione dei tradizionali linguaggi di programmazione, oppure un linguaggio specifico per basi di dati.
3. gli **Utenti finali** possono accedere alla base di dati attraverso i comandi di un linguaggio di interrogazione (query language), oppure, per utenti finali ancora meno esperti attraverso interfacce software, che presentano sul video il menu o le icone.

Il Linguaggio SQL

Il linguaggio SQL (Structured Query Language) è un linguaggio non procedurale che è diventato uno standard tra i linguaggi per la gestione di db relazionali. Dopo una prima versione introdotta da IBM alla fine degli anni 1970 per un prototipo di ricerca denominato System R, negli anni 1980 fu adottato con il nome attuale come linguaggio per i software DBMS prodotti dalla IBM (DB2 e SQL/DS).
Nel 1986 l'ANSI (American National Standards Institute) pubblicò il primo standard del linguaggio SQL, al quale seguì lo standard ISO (International Standards Organization) pubblicato nel 1987. Successivamente le specifiche del linguaggio vennero precisate e ampliate: gli aggiornamenti degli standard furono pubblicati nel 1992 da ANSI e ISO con l'SQL-92 (o SQL2, il più usato). L'SQL 1999 (o SQL3), deve ancora affermarsi e presenta delle aggiunte orientate agli oggetti.
Oggi il linguaggio SQL viene usato in tutti i prodotti DBMS come linguaggio di comandi per l'utente della base di dati (ricordiamo Oracle, Informix, SQL Server, Access).

Il linguaggio SQL consente all'utente di:

1. definire la struttura delle relazioni del database (DDL)
2. modificare i dati contenuti nel db (inserimento, variazione e cancellazione) (DML)
3. gestire il controllo degli accessi e i permessi per gli utenti (DCL)
4. interrogare il db (funzioni di Query Language).

Ogni sistema che aderisce allo standard deve fornire almeno uno dei tre possibili modi di usare SQL
1. L'uso interattivo dell' SQL;
2. SQL "ospite" di un linguaggio di programmazione;
3. SQL per scrivere procedure il cui corpo è un singolo comando SQL.

Sql per programmare le applicazioni
Uno dei principali obiettivi dei progettisti dell'sql fu che il linguaggio dovesse essere utilizzabile direttamente da utenti occasionali per interrogare basi di dati. La tendenza attuale è quella di usare sql come componente dedicata all'accesso ai dati all'interno di un linguaggio di programmazione di natura diversa. Ci sono tre categorie di linguaggi:

1) Linguaggi tradizionali
Come per esempio Pascal, C, Basic estesi con l'sql per operare sulle bd (Embedded). Presentano come vantaggio il costo ridotto di addestramento del programmatore che continuerà ad usare un linguaggio già conosciuto ma come svantaggio il fenomeno detto impedence mismatch ovverosia la differenza tra i tipi di dati del linguaggio e quelli relazionali obbliga a curare la conversione dei dati fra i due diversi modelli. Occorre un precompilatore che controlla i comandi sql, li sostituisce con chiamate a funzioni predefinite e genera un programma nel linguaggio convenzionale.

2) Linguaggi con interfaccia API (Application Program Interface)
Linguaggio convenzionale che usa delle funzioni di libreria predefinita per usare sql. I comandi sono stringhe passate come parametri alle funzioni che poi vengono controllate dinamicamente dal DBMS prima di eseguirle. C'è un SQL dinamico cioè a Runtime e il compilatore non controlla nulla. Si usa per quelle applicazioni che si prestano ad essere scritte in java ma per le quali si ha bisogno di accedere anche al db.
Invece di modificare il compilatore di un linguaggio, si usa una libreria di funzioni/oggetti (API) che operano su db alle quali si passa come parametro stringhe sql e ritornano il risultato nel quale si opera con una logica ad iteratori.

- Microsoft ODBC è C/C++ standard su Windows

- Sun JDBC è l'equivalente in Java
- Dovrebbero essere indipendenti dal DBMS:
- Un "driver" gestisce le richieste e le traduce in un codice specifico
- La db può essere in rete

3) Linguaggi Integrati (Dati e DML)
Linguaggio disegnato ad hoc per usare sql. I comandi sql sono controllati staticamente dal traduttore ed eseguiti dal dbms. Per ovviare al problema dell'impedence mismatch presente nell'uso di sql con un linguaggio ospite, si integrano i costrutti di un linguaggio relazionale e di un linguaggio di programmazione in un unico linguaggio.
Es. PL SQL / ORACLE o 4GL / INFORMIX
Permettono di :
definire variabili di tipo scalare (record annidati)
definire tipi a partire da quelli della bd
definire query sql ed esplorarne il risultato
modificare la db
definire procedure o moduli
gestire il flusso del controllo, le transazioni, le eccezioni ecc
Con questi linguaggi di solito ci si scrivono i Trigger.

SQL

L'SQL racchiude i quattro linguaggi descritti in precedenza, nella tabella che segue vengono elencati i principali comandi suddivisi per tipologia:

DDL	DML	DCL	QL
create	insert	grant	select
alter …..drop | add	update	revoke	
drop table | index	delete		

Le istruzioni per creare, modificare ed eliminare le tabelle dal database relazionale, cioè le funzioni di linguaggio DDL (Data Definition Language) sono :
La create table crea una nuova tabella nel data base attivo. La sintassi è :

CREATE TABLE nome_tabella (elenco_attributi , tipo, obbligatorietà)

create table *studenti*
(
matricola integer ,
nome char(20) ,
cognome char(30) ,
sezione char(2) ,
classe integer
);

La alter table consente di aggiungere (ADD) o cancellare (DROP) colonne dalla tabella specificata:

alter table *studenti add data_nascita date;*

alter table studenti ***drop*** *sezione;*

L’istruzione create index serve a definire un nuovo indice su una tabella esistente, se non si vogliono valori di attributi duplicati per righe diverse si deve utilizzare la clausola unique

create unique index *nome* ***on*** *studenti (cognome, nome) ;*

L’ istruzioni ***drop table*** *studenti ;* cancella la tabella *studenti*.
L’istruzione ***drop index*** *nome* ***on*** *studenti ;* cancella l’indice *nome* associato a *studenti.*

Le istruzioni per inserire, modificare ed eliminare le righe di una tabella, cioè le funzioni di linguaggio DML (Data Manipulation Language) sono:
INSERT per inserire una nuova riga in una tabella

insert into *studenti (matricola, nome ,cognome , sezione , classe)*
values *(1234, ‘Paolo’, ‘Rossi’, ‘Ai’, 5) ;*

UPDATE per aggiornare i valori nella tabella. Ad esempio pèer cambiare la classe allo studente di matricola 1234 dovremo:

update *studenti*
set *classe = 4*
where *matricola = 1234 ;*

DELETE cancella righe di una tabella. Ad esempio per cancellare lo studente di matricola 1234

delete from *studenti*
where *matricola = 1234 ;*

La select è il comando principale di SQL che realizza le funzioni di Query Language la sua struttira generale è :
select ….
from ….
where ….

dopo *select* : nomi delle colonne da elencare (* se li vogliamo tutti)
dopo *from* : il nome o i nomi delle tabelle
dopo *where* : la condizione da controllare sui valori delle righe (anche più condizioni combinate con gli operatori AND, OR e NOT).
Con ***select distinct...*** le righe duplicate nella tabella risultante vengono ridotte a una.

select *
from *studenti*
where *nome = ‘Paolo’* ***;***

crea una nuova tabella a partire dalla tabella studenti contenenete le sole righe relative agli studenti di nome Paolo (selezione).

select *nome, cognome*
from *studenti;*

crea una nuova tabella a partire dalla tabella studenti contenenete le sole colonne specificate (proiezione).

Le operazioni relazionali in SQL

La tabella che segue ci dice come realizzare in SQL gli operatori relazionali
Date le relazioni **R (a, b, c)** ed **S (a, f, g)**

	Operatore	SQL	Descrizione
Proiezione	$\pi_{a, b}(R)$	*SELECT* **a, b** *FROM* **R**	Produce una nuova tabella con le sole colonne a e b della tabella R
Selezione	$\sigma_{a = valore}(R)$	*SELECT* ***** *FROM* **R** *WHERE* a = valore	Produce una nuova tabella costituita dalle righe della tabella R per cui a = valore è vera.
Unione (*)	$R \cup S$	*SELECT* ***** *FROM* **R** *UNION* *SELECT* ***** *FROM* **S**	Produce una nuova tabella costituita dalle righe di R e dalle righe di S.
Intersezione (*)	$R \cap S$	SELECT ***** FROM **R** WHERE EXIST (SELECT ***** FROM **S** WHERE e = **R** .a AND f = **R** .b AND g = **R** . c)	Produce una nuova tabella costituita dalle righe che appartengono sia ad R che ad S.
Differenza (*)	$R - S$	SELECT ***** FROM **R** WHERE NOT EXIST (SELECT ***** FROM **S** WHERE e = **R** .a AND f = **R** .b AND g = **R** . c)	Produce una nuova tabella costituita dalle righe di R che non sono anche righe di S.
Prodotto Cartesiano	$R \times S$	SELECT ***** FROM **R, S**	Produce una nuova tabella costituita dalle righe di R concatenate alle righe di S.
Giunzione (**)	$R \mid\times\mid_{R.a = S.a} S$	SELECT ***** FROM **R, S** WHERE **R** . a = **S** . a	Produce una nuova tabella ottenuta : $\sigma_{R.a = S.a}(R \times S)$

(*) Le relazioni devono avere lo stesso grado e gli attributi di stessa posizione devono essere dello stesso tipo(relazioni congruenti).
(**) Le relazioni devono avere un attributo comune.

Le funzioni di aggregazione

sono funzioni predefinite che agiscono sui valori contenuti in insiemi di righe della tabella e restituiscono un valore calcolato.
La funzione **COUNT** restituisce il numero di righe presenti in una tabella.

select count (*) from nome_tabella
select count (attributi) from nome_tabella

numero di studenti di nome paolo
select count(*) from studenti
where nome = ‘ paolo’ ; restituisce il numero di studenti di nome paolo

La funzione **SUM** restituisce la somma di tutti i valori contenuti in una colonna specificata (l’attributo deve essere di tipo numerico).

select sum (attributo) from nome_tabella ;

Data la relazione m*ovimento (numero, descrizione, data, importo, codice)*
Select sum(importo) from movimento restituisce il totale dei movimenti

La funzione **AVG** calcola la media (average) dei valori (numerici) contenuti in una determinata colonna di una tabella.

select avg (attributo) from nome_tabella ;

s*elect avg (Importo) from* movimento ; restituisce l’importo medio dei movimenti

Le funzioni **MIN e MAX r**estituiscono rispettivamente il valore minimo e il valore massimo tra i valori della colonna specificata come argomento della funzione (anche per campi di tipo carattere).

Select Min (attributo), *Max* (attributo *) from* nome_tabella

Ordinamenti
La clausola ORDER BY consente di ottenere i risultati di un’interrogazione ordinati secondo i valori contenuti in una o più colonne, tra quelle elencate accanto alla parola Select.
SELECT colonna1, colonna2
FROM nome_tabella
ORDER BY colonna1
L’ ordinamento di default è quello crescente (**ASC**) se vogliamo il decrescente occorre la clausola **DESC**

select (*) from studenti
order by cognome restituisce l’elenco degli studenti ordinato in ordine crescente

Raggruppamenti

La clausola GROUP BY serve per raggruppare un insieme di righe aventi lo stesso valore nelle colonne indicate: produce una riga di risultati per ogni raggruppamento.
Viene usata con le funzioni di aggregazione (Sum, Count...) : per ciascuna riga della tabella risultante viene prodotto un valore di raggruppamento.

SELECT Colonna, Funzione
FROM NomeTabella
GROUP BY Colonna

Esempio:
Data la tabella:
Movimento (Numero, Descrizione, Data, Importo, Codice)
Totale degli importi dei movimenti per ciascun codice anagrafico
Select Codice, Sum(Importo)
From Movimento
Group By Codice

Condizioni sui raggruppamenti

L'uso della clausola HAVING consente di sottoporre al controllo di una o più condizioni i gruppi creati con la clausola Group by.
La condizione scritta dopo Having normalmente controlla il valore restituito dalle funzioni di aggregazione (Count, Sum, Avg, Min, Max).

SELECT colonna, funzione
FROM nome_tabella
GROUP BY colonna
HAVING condizione

Data la tabella *movimento (numero, descrizione, data, importo, codice)*
Importo medio dei movimenti per i codici aventi più di 20 movimenti registrati

Select Codice, Avg (Importo)
From Movimento
Group By Codice
Having Count(*) > 20

Attenzione alla differenza tra
Where che pone condizioni sulle righe della tabella
Having che controlla condizioni su gruppi di righe

Le condizioni di ricerca

Il linguaggio SQL utilizza operatori e predicati insieme alle clausole *Where* e *Having* per determinare i criteri di selezione rispettivamente delle righe e dei raggruppamenti.

- Segni del confronto =, <, >, <>, >=, <=.
- Più condizioni legate tra loro con gli operatori AND, OR e NOT.

BETWEEN controlla se un valore è compreso all'interno di un intervallo di valori, inclusi gli estremi.

Elenco dei movimenti con importo compreso tra 100 e 200
Select *
from movimento

where importo *between* 100 *and* 200

IN controlla le righe che hanno i valori di un attributo compresi in una lista di valori indicati dopo la parola in.

select *
from studenti
Where nome *IN* ('Marco', 'Paolo')
restituisce le righe che hanno l'attributo nome uguale a Paolo e Marco.
LIKE confronta il valore di un attributo di tipo carattere con un modello di stringa che può contenere caratteri jolly :
_ (underscore) per indicare un singolo carattere qualsiasi in quella posizione della stringa;
% (percento) per indicare una sequenza qualsiasi di caratteri in quella posizione della stringa.
Per esempio:
LIKE 'xyz%' vengono ricercate tutte le stringhe che iniziano con i caratteri 'xyz' ;
LIKE '%xyz' serve a ricercare tutte le stringhe che finiscono con i caratteri 'xyz' ;
LIKE '%xyz%'' per tutte le stringhe che contengono al loro interno i caratteri 'xyz';
LIKE '_xyz' controlla le stringhe di 4 caratteri che finiscono con xyz.

Elenco dei cognomi che iniziano con 'Ros' (Rossi, Rosi, Rossini,...)
select *
from studenti
where cognome *Like* 'Ros%' ;

I comandi per la sicurezza
Abbiamo visto che il DCL (Data Control Language) linguaggio oltre che per per fissare i vincoli di integrità, serve anche per stabilire le autorizzazioni agli accessi e i tipi di permessi consentiti agli utenti (inserimento di nuovi dati, solo lettura, modifica dei dati). In SQL i comandi sono :
GRANT che consente determinate operazioni su una tabella a determinati utenti.

grant update on studenti *to* utente

REVOKE che revoca determinate operazioni su una tabella a determinati utenti.

revoke update on studenti *to* utente

Le operazioni possono essere: ***alter, delete, index, insert, select, update,all***
I permetti con le operazioni select e update diventano più restrittivi specificando, tra parentesi tonde l'elenco degli attributi che può vedere o modificare.
grant update (livello, stipendiobase) *on* personale *to* utente ;

Le viste
L'SQL consente le modalità con le quali gli utenti possono vedere le tabelle del data base, cfeando una finestra ; detta **VIEW**, su alcuni o tutti i dati contenuti in una o più tabelle.
Alla vista viene assegnato un nome in fase di creazione con il comando **CREATE VIEW** . Con le viste è possibile decidere ciò che l'utente debba 'vedere' di una tabella tabella, crando così maggiore sicurezza. Nella viste è possibile eseguire tutte le operazioni viste sino ad ora tranne in quelle che sono risultato di funzioni di aggregazione.

create view studentiV
as select *

from studenti
where classe = V ;

drop view studentiV cancella la vista.

Interrogazioni nidificate

Una interrogazione nidificata è una interrogazione che contiene un'altra interrogazione. SQL non pone limiti al livello di annidamento ma solitamente ci si ferma a due. E' possibile nidificare una interrogazione nella clausola ***where*** (vedremo un'altra forma di annidamento parlando delle viste). In particolare una espressione può essere confrontata mediante gli usuali operatori di confronto con una interrogazione. L'operatore di confronto è seguito dalla parola chiave *any* oppure *all*. Nel primo caso, il predicato è vero se ha successo il confronto (secondo l'operatore usato) tra il valore dell'espressione e almeno uno tra i valori risultato della query. Nel secondo caso, il predicato è vero se ha successo il confronto (secondo l'operatore usato) tra il valore dell'espressione e tutti i valori risultato della query. L'espressione e il risultato dell'interrogazione devono essere compatibili, cioè avere lo stesso numero di attributi e gli attributi devono evere domini compatibili.

Vediamo alcuni esempi. Consideriamo le tabelle

attore

nome	dataDiNascita	luogoDiNascita
Elisa Bottega	1972-04-29	Treviso
Lorenzo Vignando	1970-05-29	Venezia
Barbara Altissimo	1982-03-01	Torino

autore

nome	dataDiNascita	luogoDiNascita
Vittorio Corte	1969-11-29	Napoli
Lorenzo Vigna	1970-05-29	Venezia
Antinisca Divito	1972-08-01	L'Aquila

La seguente interrogazione seleziona tutti gli attori che sono anche autori (intersezione).

```
select nome
from attore
where nome = any ( select nome
                   from autore );
```

L’ interrogazione che segue seleziona tutti gli attori che non sono autori. Si noti che abbiamo già scritto la stessa query usando l'operatore `except`:

```
select nome
from attore
where nome <> all ( select nome
                    from autore )
```

Selezionare tutti gli attori o autori è invece possibile solo usando l'operatore `union`. Le combinazioni `= any` e `<> all` sono molto frequenti e hanno un nome: nel primo caso posso usare la parola chiave `in` e nel secondo `not in`. Ad esempio:

```
select nome
from attore
where nome not in ( select nome
                    from autore )
```

Se il nome e il cognome di un attore fosse stato specificanto usando due attributi invece che uno soltato, avremmo potuto usare il costruttore di tupla `(nome, cognome)` in questo modo:

```
select nome, cognome
from attore
where (nome, cognome) not in ( select nome, cognome
                               from autore )
```

Il nome del dipendente con lo stipendio massimo può essere estratto nei seguenti due modi:

```
select nome, stipendio
from dipendente
where stipendio >= all (select stipendio
                        from dipendente)
select nome, stipendio
from dipendente
where stipendio = (select max(stipendio)
                   from dipendente)
```

Si noti che nel secondo caso la parola chiave `all` o `any` può essere omessa in quanto la query interna restituisce un solo elemento.

L'interrogazione interna può essere sostituita con un **insieme esplicito**. L'interrogazione seguente recupera tutti i nomi dei dipendenti in un certo insieme esplicito:

```
select nome
from dipendente
where nome in ("Elisa Bottega", "Lorenzo Vignando", "Barbara Altissimo")
```

Le interrogazioni nidificate viste fin ora possono essere risolte indipendentemente dall'interrogazione che le contiene. Ad esempio, nell'ultima interrogazione scritta, è possibile prima di tutto risolvere la query interna, cioè calcolare il massimo stipendio, e poi elaborare la query esterna confrontando il massimo stipendio con il valore dello stipendio di ogni dipendente. Questa soluzione è più efficiente rispetto a quella che valuta la query interna per ogni tupla della query esterna. Questa soluzione non può però sempre essere seguita. Consideriamo una interrogazione che restituisce gli attori per i quali esiste un altro attore con la medesima data di nascita. Osserviamo innanzitutto che è possibile risolvere l'interrogazione con il seguente join:

```
select distinct A1.nome
from attore A1 join attore A2 on
(A1.dataDiNascita = A2.dataDiNascita) and
(A1.nome <> A2.nome)
```

Vediamo ora una soluzione che usa una query nidificata e l'operatore `exists`. Questo operatore ha come argomento una query interna e restituisce vero se e soltanto se il risultato della query argomento contiene qualche elemento:

```
select A1.nome
from attore A1
where exists (select A2.nome
              from attore A2
              where (A1.dataDiNascita = A2.dataDiNascita) and
                    (A1.nome <> A2.nome))
```

Questo tipo di query si dicono **query nidificate con passaggio di variabili**. In particolare, la variabile A1, creata nella query esterna per la prima istanza della tabella attore, viene passata alla query interna che ne fa uso. In tal caso la query interna non può essere valutata indipendentemente da quella esterna in quanto si serve di variabili definite a livello di query esterna. Dunque l'unico modo di risolvere questa query consiste nel valutare la query interna per ogni tupla della query esterna. La **visibilità delle variabili** in SQL segue la seguente semplice regola: una variabile è visibile nella query che l'ha definita o in una query nidificata in essa (ad un qualsiasi livello).

Vediamo un esempio che usa la negazione dell'operatore `exists`: gli attori per i quali non esiste un altro attore con la medesima data di nascita:

```
select A1.nome
from attore A1
where not exists (select A2.nome
                  from attore A2
                  where (A1.dataDiNascita = A2.dataDiNascita) and
                        (A1.nome <> A2.nome))
```

Una formulazione che non usa le query nidificate è la seguente:

```
select nome
from attore
  except
select A1.nome
from attore A1 join attore A2 on
(A1.dataDiNascita = A2.dataDiNascita) and
(A1.nome <> A2.nome)
```

Una formulazione che non usa le query nidificate è la seguente:

```
select nome
from attore
  except
select A1.nome
from attore A1 join attore A2 on
(A1.dataDiNascita = A2.dataDiNascita) and
(A1.nome <> A2.nome)
```

INTRODUZIONE ALL' HTML

Per iniziare a scrivere una pagina HTML serve editor ad esempio Notepad.
L'HTML ha alcune regole:

- i ritorni a capo devono essere indicati esplicitamente con il tag
.
- il browser trascura tutti i comandi che non riesce ad interpretare correttamente.
- il browser non tiene conto degli spazi in più.
- La pagina dovrà esser salvata con estensione ***.html*** e per visualizzarla con il browser dovrete eseguire "File/Apri".

La struttura generale della pagina HTML è :

```
<HTML>
 <HEAD>
    <TITLE>Qui ci va il Titolo della finestra</TITLE>
 </HEAD>
<BODY>
 ... testo, immagini, link comandi di formattazione ecc.ecc del documento ...
</BODY>
</HTML>
```

Ogni documento HTML comincia con il tag <HTML> e finisce con <\HTML>
Ogni documento è composto da:
Body contiene il testo del documento stesso insieme ai comandi di formattazione
Head contiene le informazioni riguardanti l'intero documento.
Nalla HEAD possono essere inseriti:
<BASEFONT ...>: permette di definire un font di base, con i vari attributi FACE, COLOR, SIZE, ecc
<BASE TARGET="nome frame">: per definire quale deve essere la finestra di default per tutti i collegamenti della pagina, a meno di dichiarazioni esplicite diverse. Vedremo meglio nel capitolo dedicato ai frames.
<BASE HREF="...">: serve per definire la directory di riferimento da cui dovranno partire tutti i collegamenti relativi contenuti nella pagina. Vedremo nel capitolo dedicato alle ancore.
<STYLE...>: serve per definire delle impostazioni stilistiche da adottare all'interno della pagina.
<META>: serve per dare delle indicazioni sul codice, come ad esempio la versione di HTML utilizzata, il software usato, parole chiavi ad uso dei motori di ricerca di internet, ecc.

Modifica degli attributi del testo
I comandi per la gestione degli attributi del testo hanno tutti la stessa forma:

- **<….>** devono essere aperti prima della porzione di testo cui l'attributo si applica
- **</...>** e vanno chiusi alla fine della porzione stessa.

Gli attributi specificabili riguardano la dimensione del testo, il grassetto, il corsivo ed alcune modalità speciali in grado di simulare, ad esempio, la resa di una macchina da scrivere. Quella che segue è una lista di alcuni attributi:
<B> Per un testo in GRASSETTO (o anche "bold") **</B>**
<I> Per un testo in CORSIVO**</I>**
<STRONG> Per utilizzare uno stile più MARCATO**</STRONG>**
<PRE> Testo PREFORMATTATO(considera tutti gli spazi ed i ritorni a capo inseriti)</PRE>

<TT> Stile da MACCHINA DA SCRIVERE**</TT>**
<**H6**> INTESTAZIONE (o "heading") di livello 6 **</H6>**
<H5> INTESTAZIONE di livello 5 **</H5>**
<H4> INTESTAZIONE di livello 4 **</H4>**
<H3> INTESTAZIONE di livello 3 **</H3>**
<H2> INTESTAZIONE di livello 2 **</H2>**
<H1> INTESTAZIONE di livello 1 **</H1>**
Al posto del tag <H…> si può utilizzare:
<FONT SIZE=7> Dimensione 7 **</FONT>**
<FONT SIZE=6> Dimensione 6 **</FONT>**
<FONT SIZE=5> Dimensione 5 **</FONT>**
<FONT SIZE=4> Dimensione 4 **</FONT>**
<FONT SIZE=3> Dimensione 3 (Normale) **</FONT>**
<FONT FACE='arial' SIZE=3 COLOR=#FF0000" > testo di colore rosso, font arial dimensione normale **</FONT>**
Nell'ultimo esempio sono stati introdotti i comandi FACE e COLOR il cui significato è evidente.

Proviamo ad inserire il testo che segue ed a salvarlo come *prova.html*

```
<HTML>
<HEAD>
  <TITLE>Test numero 2</TITLE>
</HEAD>
<BODY>
  <H1>esempio di un documento HTML.</H1> <BR>
  <I>esempio testo in Corsivo.</I><BR>
  Questo è un testo Normale. <BR>
  <TT>esto è stato scritto da una macchina da scrivere!</TT><BR>
  <H3><I>uso contemporaneo di attributi diversi?</H3></I><BR>
 </BODY>
</HTML>
```

Sono disponibili molti altri comandi ma la loro descrizione non è negli scopi di questi appunti.

Formattazione dei paragrafi

I browser, come già detto, non tengono conto della formattazione del testo in HTML, cioè non tengono conto di spaziatura ridondanti, ritorno a capo, tabulazioni, ecc. Questo perchè la pagina deve poter essere vista con qualunque browser e in finestre di dimensione diversa. Sarà dunque il browser stesso a riformattare la pagina.

Il trasferimento del codice HTML avviene con 7 bit per carattere: questo significa che vengono trasmessi solo i primi 128 caratteri del codice ASCII. Per visualizzare gli altri caratteri, come lettere accentate, con dieresi, ecc., si usano dei codici particolari.

Abbiamo già visto il tag **
** che consente di andare a capo. Per imporre ad una porzione di testo di non andare mai a capo, la si deve racchiudere fra **<NOBR> e </NOBR>.**

Per cambiare paragrafo, invece, si usa la tag **<P>** e **</P>** che oltre a mandare a capo, lascia anche una riga di spazio.

Il tag **<HR>** consente la tracciatura di una linea orizzontale. Il browser dimensionerà automaticamente la riga sulla larghezza della finestra che si stà utilizzando. Questa tag ha degli qualche attributo. Essi sono:

SIZE = valore specifica lo spessore della riga in pixel.
WIDTH = valore lunghezza inpixel o in percentuale (%) rispetto alla larghezza della finestra
N**OSHADE** : impone che la riga sia solida e non un solo effetto di ombra.

ALIGN = "tipo": tipo di allineamento, che può essere LEFT, RIGHT, o CENTER.
Ad esempio l'istruzione: <HR ALIGN="RIGHT" WIDTH="50%" SIZE=5> .
Altro comando di formattazione dei paragrafi è **<BLOCKQUOTE> ... </BLOCKQUOTE>.** Tutto il testo compreso fra tali tag, verrà visualizzato leggermente rientrato ai lati;

Inserire immagini

L'HTML prevede il tag **<IMG>** per inserire immagini, è un tag aperto, cioè quindi non necessita della chiusura <\...>. <IMG> prevede vari parametri:

SRC per indicare l'immagine da inserire
LOWSRC per indicare un'altra immagine da caricare, prima della definitiva.
BORDER per indicare la dimensione dell'eventuale bordo intorno all'immagine.
ALIGN per indicare l'allineamento dell'immagine rispetto al testo. Può assumere i valori LEFT, RIGHT o CENTER.
ISMAP indica che l'immagine visualizzata è una mappa di coordinate: clickandoci sopra il computer rileva le coordinate del mouse ed è in grado di inviarle al server che può elaborarle per individuare, ad esempio, una nuova pagina da inviare.
WIDTH specifica una larghezza predefinita per l'immagine, con un valore dato in punti.
HEIGHT analogamente al precedente imposta un'altezza predefinita.
VSPACE per indicare uno spazio da lasciare libero sopra e sotto l'immagine.
HSPACE per indicare uno spazio da lasciare libero a destra e a sinistra dell'immagine.
ALT visualizza un testo al posto dell'immagine se questa non è visualizzabile.
USEMAP per utilizzare, sull'immagine, una mappa locale.

ESEMPIO: < IMG SRC = "imagine.jpg" ALIGN = "RIGHT" >
inserisce un'immagine allineata a destra. I formati dell'immagine usabili sono tanti ma quelli universalmente riconosciuti sono il GIF ed il JPEG.

< IMG SRC="image1.jpg" BORDER = 10 WIDTH = 180 HEIGHT = 80>

che la formattano a dimensioni obbligate di 180x80 punti mostrando un bordo di 10 punti intorno. Se non viene specificato altrimenti tramite il desiderato allineamento, l'immagine viene inserita nel testo esattamente nel punto in cui appare il comando IMG, disponendo il testo ai suoi lati.
Il parametro **ISMAP** permette di indicare al browser di attivare la modalità di "tracking" del mouse: *quando si preme il pulsante vengono rilevate le coordinate ed inviate al server*. Questo, dotato di un apposito programmino, è in grado di gestire le coordinate ricevute e di comportarsi di conseguenza.
ALT nel caso venga utilizzato un browser testuale, indica il testo alternativo all'immagine ad esempio, < IMG SRC = "image.gif" ALT = "Immagine 1" >
Le immagini possono naturalmente essere inserite in tabelle così da formattarne la disposizione a piacimento.
I' **ALIGN=""** aggiunge effetti particolari. Ad esempio se vogliamo di inserire un testo al centro di due immagini, disposte ai lati della finestra :

```
<IMG SRC="image1.jpg" ALIGN="RIGHT" WIDTH="100">
<IMG SRC="image1.jpg" ALIGN="LEFT" WIDTH="100">
<CENTER>TESTO ALLINEATO AL CENTRO DELLE DUE IMMAGINI</CENTER>
<BR CLEAR="RIGHT">
<BR CLEAR="LEFT">
```


TESTO ALLINEATO AL CENTRO DELLE DUE IMMAGINI

Quando si specifica un comando ALIGN tutto ciò che segue il tag <IMG ...> viene posto accanto all'immagine fino al riempimento dello spazio a disposizione. Se non volete che ciò accada dovete usare il tag <BR CLEAR="..."> che cancella l'allineamento specificato nell'<IMG ...>. Nell'esempio appena visto volevamo inserire tra le due immagini SOLO il testo desiderato e lasciare il rimanente spazio vuoto. Abbiamo dovuto inserire i due tag <BR CLEAR="RIGHT"> e < BR CLEAR="LEFT"> subito dopo il testo per far si che tutto ciò che segue sia inserito FUORI dello spazio tra le due immagini.

Le immagini possono essere inserite all'interno di comandi <A HREF="..."> per utilizzarle come pulsanti di collegamento od icone. Il browser, se non altrimenti specificato, mette automaticamente un bordo intorno all'immagine per indicare che è un link. Se non lo volete usate il parametro BORDER=0 dentro IMG come:

```
<A HREF = "link.htm">
<IMG SRC = "image1.jpg" BORDER = 0>
</A>
```

I collegamenti

Per rendere “cliccabili” un oggetto si utilizzano le *ancore*. Un'ancora non è altro che un riferimento all'indirizzo di una pagina WWW. Per crearla il comando è:

< A HREF = "URL" > ..... </A>

dove URL stà per Uniform Resource Locator, ovvero l'indirizzo della risorsa (una pagina, un’ immagine ecc..ecc..).
Un URL è composto da:

- servizio richiesto (http, ftp, mailto ..);
- nome del dominio che fornisce la risorsa;
- numero della porta utilizzata sul server (se manca è 80)
- percorso (path) per arrivare alla risorsa;
- eventuale **ancora interna** al documento.

ESEMPIO http://www.sito.it/cartella/documento.html

Tutto ciò che si trova fra le tag di apertura e chiusura di un'ancora (testo o immagini diventa "cliccabile".

Il browser provvederà ad evidenziare le ancore, per renderle di facile individuazione, con sottolineature e/o colori diversi per il testo e contornando le figure. Se col cursore si passa sopra un'ancora, esso diventerà una manina, a sottolineare il fatto che quello è un collegamento.
Se il documento da linkare è nella stessa directory della pagina attualmente visualizzata, non è necessario dare l'URL, ma è sufficiente scrivere il nome del file: < A HREF = "pagina.htm" >

L’ ancora interna non è altro che un segnalibro all’interno della pagina. In tal caso non si dovrà utilizzare un URL come riferimento, ma semplicemente un nome, preceduto da un cancelletto (#). Il segnalibro sarà costituito dall’ attributo NAME="....".

```
…….
<BODY>
        …….
        < A HREF = "#fine" >Clicca qui per andare alla fine del documento</A>
        ……
        ……
       < A NAME="fine" >
       ……
       ……
<BODY>
……..
```

Nell’esempio che segue i due tipi di ancore coesistono:
<A HREF = "pippo.htm#pluto">
Viene caricata la pagina pippo.htm (che è nella stessa cartella) , a partire dal punto in cui si trova l'ancora interna pluto.
Le ancore possono anche collegarsi ad immagini, suoni, animazioni, ecc. Ad esempio:

```
<A HREF="immagine.gif">
<A HREF="suono.wav">
<A HREF="filmato.mov">
```

A seguito del click su un riferimento del genere, il browser caricherà il file e lo manderà in esecuzione. Possiamo creare un’ancora per richiamare il client di posta predefinito sul nostro sistema e quindi inviare un’ email:

```
<A HREF = "mailto : pippo@pluto.it">clicca qui per inviarmi una email</A>
```

Le liste

Una lista è composta da una serie di paragrafi, ciascuno dei quali viene identificato dal comando **<LI>** e che possono essere elencati ordinati, non ordinati o visualizzati su colonne multiple.
Per specificare il tipo di lista dobbiamo inserire una delle seguenti istruzioni:

<OL> ... </OL> per una lista ordinata e numerata.
<UL> ... </UL> per una lista non ordinata
<MENU> ... </MENU> per una lista di brevi voci. In genere occupa una riga e viene visualizzato in modo più compatto dell'UL.
<DIR> ... </DIR> per visualizzare un tipico elenco di nomi di file.

Per inserire una lista è allora sufficiente specificare il tipo di lista richiesta (OL, UL, MENU o DIR) e poi inserire ogni elemento tramite il comando <LI>.

```
<HTML>
 <HEAD> <TITLE>Prova di una lista ordinata </TITLE></HEAD>
 <BODY>
```

```
 <H1>Indice di un libro HTML</H1>
 <H3>
  <OL>
     <LI> Introduzione all'HTML
     <LI> Specifiche del linguaggio
     <LI> Come scrivere un documento
   </OL>
 </BODY>
</HTML>
```

Indice di un libro HTML

1. Introduzione all'HTML
2. Specifiche del linguaggio
3. Come scrivere un documento

Si noti che non esiste un identificatore di fine elemento (</LI>). OL, UL, MENU e DIR devono esser chiusi da </…>.

Esiste l'attributo "TYPE" che serve per indicare il tipo di indicizzazione che si vuole avere sulla lista. Ad esempio, il comando
< OL TYPE = "a" >, creerà una lista ordinata, utilizzando le lettere minuscole come indice. Gli altri valori possibili sono:
"A" per usare lettere maiuscole;
"1" (default) per utilizzare i numeri;
"I" per usare la numerazione romana maiuscola;
"i" per i numeri romani minuscoli.
Per le liste non ordinate i possibili valori sono:
"DISC" per dei cerchi pieni;
"CIRCLE" per dei cerchi vuoti;
"SQUARE" per dei quadratini .

Per le liste <OL>, infine, esiste un ulteriore attributo, **START**, per indicare il punto da cui far partire la numerazione: ad esempio *<OL TYPE="A" START="5">* farà partire la numerazione dalla lettera "E". Naturalmente questo attributo non ha senso per le liste non ordinate.

Le Tabelle

Una tabella può essere inserita molto facilmente in una pagina semplicemente componendola tramite un editor ed inserendola con il comando <PRE>, che ne conserva intatta la formattazione:

```
<HTML>
<HEAD> <TITLE>Una  tabella</TITLE> </HEAD>
<BODY>
 <PRE>
        <FONT SIZE=2>
        Andamento prezzi trimestrale
        Componente Gennaio Febbraio Marzo
        ----------------------------------------
        CPU 250.000 235.000 198.000
        Hard Disk 470.000 436.000 421.000
```

```
        </FONT>
  </PRE>
</BODY>
</HTML>
```

L'aspetto grafico di questa tabella può essere notevolmente migliorato utilizzando il comando: **<TABLE>.** Questo comando indica il punto d'inizio dei dati che vanno inseriti nella tabella e prevede il tag di chiiusura. Accetta inoltre un parametro **BORDER** che specifica lo spessore del bordo della tabella.

Vediamo allora come si può implementare una tabella tramite il comando <TABLE>

```
<table border="1">
   <tr>
      <td>prima cella</td>
      <td>seconda cella</td>
   </tr>

   <tr>
      <td>terza cella</td>
      <td>quarta cella</td>
   </tr>
</table>
```

Che viene visualizzato così:

prima cella	seconda cella
terza cella	quarta cella

Ancora un altro esempio:

```
<table border="1"  width="400">
  <tr>
     <td>TutorialPc</td>
     <td>La guida</td>
  </tr>
  <tr>
     <td>All'informatica</td>
     <td>Qui impari</td>
  </tr>
  </table>
```

Ecco il risultato che si ottiene

TutorialPc	La guida
All'informatica	Qui impari

Vediamo ora più in dettaglio la funzionalità di ogni comando:
CAPTION …..</CAPTION>. Consente di inserire il titolo della tabella.
<TR>…</TR> racchide una nuova RIGA della tabella. Va inserito ad ogni cambio riga.
<TH>…</TH> intestazione di una nuova colonna.
<TD>…</TD> Nuova casella DATI. Specificando in sequenza vari comandi <TD> con i relativi dati si introducono consecutivamente tutte le celle di una stessa riga. Per passare alla riga successiva si usi il comando <TR>.

Il comando TABLE può avere i seguenti attributi:

WIDTH definisce la larghezza della tabella, in pixel o in percentuale rispetto alla finestra del browser.
HEIGHT Specifica l'altezza della tabella.
ALIGN Specifica l'allineamento del contenuto delle celle.
VALIGN **S**pecifica l'allineamento verticale rispetto ai bordi delle celle.
BGCOLOR Definisce, con le solite modalità (BGCOLOR="#rrggbb"), il colore di sfondo della tabella.
BORDERCOLOR Definisce il colore dei bordi della tabella.
BORDER Specifica la larghezza del bordo esterno della tabella; "0" per non avere bordi.
CELLPADDING Serve per imporre un contorno interno alle celle che deve essere vuoto.
CELLSPACING Impone la distanza fra le celle, ovvero la larghezza dei bordi interni.

I comandi <TR>, <TH> e <TD> possono comprendere una serie di parametri:

ALIGN Permette di specificare l'allineamento del contenuto di una cella. Può assumere i valori RIGHT, LEFT e CENTER. Se utilizzato con il comando <TR> permette di associare un allineamento a tutti gli elementi di una stessa riga. Il default è LEFT per le celle dati e CENTER per le celle d'intestazione.
VALIGN Funziona esattamente come ALIGN, ma permette di stabilire l'allineamento verticale dei paragrafi nelle celle. I valori accettati sono TOP (allineamento in alto), MIDDLE (centratura) e BASELINE (allineamento in basso). Il default è CENTER per tutte le celle.
COLSPAN Controlla il numero di colonne su cui una cella si può estendere. Utilizzandolo insieme a ROWSPAN è possibile creare delle celle larghe 2 o piu' righe/colonne.
ROWSPAN Definisce il numero di righe su cui incide la casella.
HEIGHT Definisce l'altezza della cella, o dell'intera tabella.
WIDTH Definisce la larghezza della cella, o dell'intera tabella.
NOWRAP Indica che il testo contenuto nella cella non dovrà essere spezzato su più linee.
BGCOLOR Specifica quale deve essere il colore dell'intera tabella, o della riga, o della singola cella.
BORDERCOLOR Specifica il colore che devono avere i bordi.
BACKGROUND Attributo riconosciuto solo da Internet Explorer, serve per specificare una immagine da utilizzare come sfondo.

Le celle della tabella possono contenere un qualsiasi elemento HTML: testo, immagini, collegamenti. Utilizzando la tabella è quindi possibile, ad esempio, creare una griglia di pulsanti come quella utilizzata in questa guida per l'indice generale, in cui sono inserite immagini accanto al testo, ed il tutto rappresenta un collegamento all'opportuna sezione.

Un'altra cosa interessante, è la possibilità di nidificare le TABLE: una cella può contenere a sua volta un'altra TABLE.

I FORM

I form sono elementi importanti per ogni sito web che vuole interagire con l'utente che ad esempio può mandare una propria opinione, chiedere informazioni,rispondere a sondaggi ecc. senza la necessità di scrivere via e-mail.
Da sottolineare che tutto ciò è possibile grazie a dei programmi che risiedono sul server che si chiamano CGI (Common Gateway Interface) quindi perchè un form Html funzioni è indispensabile un programma CGI residente sul proprio server.

I tag **<FORM>…….</FORM>** aprono e chiudono il form.

****

Adesso analizziamo questa sintassi:

< FORM method = "get|post" action = "http://www.tuosito.com/cgi-bin/nome_script.cgi" >

Se **medod = "get"** ciascun campo viene inviato al server come una coppia attibuto-valore. Ha lo svantaggio che i dati sono inviati in chiaro insieme all'indirizzo e che la stringa non può superare i 255 caratteri.
Se **method="post"** i dati non sono visibili e non vi è limite di lunghezza.

TEXT
Il valore TEXT crea i campi di testo, dove vengono richiesti dati quali il nome o l'indirizzo e-mail. E' un valore usato soprattutto per informazioni non predefinite che variano di volta in volta e con i seguenti attributi si crea un campo testo:

maxlength definisce il numero massimo di caratteri inseribili nel campo.
size: definisce la larghezza della stringa all'interno della pagina .
value: visualizza un testo di default all'interno della stringa.

Esempio:
<INPUT type="TEXT" name="nome" maxlength="30" size="20" value="Inserisci nome">

ESEMPIO

inserisci nome

PASSWORD
Questo campo ha una funziona simile a TEXT visto in precedenza, ma con la differenza che quando si digita all'interno della stringa bianca, non appaiono le lettere ma i classici asterischi delle password (anche se in realtà i dati non vengono codificati e quindi non c'è nessuna sicurezza).
ESEMPIO

<INPUT type="PASSWORD" name="nome" maxlength="30" size="20">

****

CHECKBOX
Questo attributo viene utilizzato per informazioni del tipo "si/no" oppure "vero/falso". Crea delle piccole caselle quadrate da spuntare o da lasciare in bianco. Se la casella e' spuntata input restituisce un valore al CGI, al contrario non restituisce alcun valore.
Value impostato su "yes" significa che di default la casella e' spuntata. Checked controlla lo stato iniziale della casella, all'atto del caricamento della pagina.

<INPUT type = "CHECKBOX" name = "età" value = "yes" checked>

ESEMPIO

RADIO

Questo attributo ha funzioni simili a CHECKBOX, ma presenta più scelte possibili. Selezionando una voce tra quelle presenti, qualora abbiano tutte valore "name" identico, si deselezionano automaticamente le altre.

<INPUT type="RADIO" name="voto" value="sufficiente">
<INPUT type="RADIO" name="voto" value="buono">
<INPUT type="RADIO" name="voto" value="ottimo">

SUFFICIENTE ○

BUONO ○

OTTIMO ○

SUBMIT
Questo è il classico bottone che invia il form con tutti i suoi contenuti. La larghezza del bottone dipende dalla lunghezza del testo.

<INPUT type="SUBMIT" value="Invia">

RESET
Bottone che cancella l'intero form eliminando i dati inseriti.

<INPUT type="RESET" value="Cancella">

TEXTAREA
Textarea viene utilizzato quando serve spazio per inserire molto testo. La larghezza e' impostata da "cols" e l'altezza da"rows".WRAP="physical" stabilisce che qualora il testo inserito superi la larghezza della finestra, venga automaticamente riportato a capo.

<TEXTAREA cols = 50 rows = 4 WRAP = "physical" name = "commento"></textarea>
ESEMPIO

HIDDEN
Serve per un campo nascosto, cioè un campo che non compare effettivamente sulla pagina. Può essere comodo per inviare dei valori all script CGI non visibili all'utente che utilizza il form.
<INPUT type ="HIDDEN" value = "valore_nascosto">

SELECT
Il comando SELECT presenta una lista di possibili scelte in un pop-up menù o in una lista a scorrimento. Le voci selezionabili compaiono premendo sulla freccia in basso.

<SELECT size=1 cols=4 NAME="giudizio">
<OPTION selected Value=nessuna>
<OPTION value = buono> Buono
<OPTION value = sufficiente> Sufficiente

<OPTION Value = ottimo> Ottimo
</SELECT>
I suoi parametri sono:

NAME E' la variabile in cui sarà memorizzato il valore scelto.

SIZE Definisce quante scelte devono essere mostrate. Se viene omesso allora si utilizza un pop-up menù. Se è settato a 2 o più, allora si utilizza una lista scorrevole. Se il numero di voci è inferiore a quanto qui specificato vengono introdotti automaticamente dei campi "Nothing".

MULTIPLE Permette di selezionare scelte multiple. Viene utilizzata comunque una lista scorrevole senza tener conto del valore di SIZE.

Le voci tra cui è possibile scegliere compaiono tra <SELECT> e </SELECT> utilizzando il comando **<OPTION>** che ha i seguenti parametri:

VALUE E' ciò che varrà trasmesso al server se la voce sarà selezionata.

SELECTED scelta che deve essere selezionata per default.

```
<HTML>
  <HEAD>       <TITLE>I form in HTML</TITLE>  </HEAD>
 <BODY>
<FORM Method="POST"  Action= "http://www.december.com/cgi-bin/formmail.secure.cgi">

<INPUT Type="hidden" Name="recipient" Value="nobody@december.com">

<p>Cognome e Nome :  <INPUT Type = "text"   size="30" Name="nome-utente">
<P>Codice Articolo : <INPUT Type = "text"   size="10" Name= "codice">

<P>Taglia della maglia?
<INPUT Type="radio" Name="taglia" Value="S">S
<INPUT Type="radio" Name="taglia" Value="M">M
<INPUT Type="radio" Name="taglia" Value="L">L
<INPUT Type="radio" Name="taglia" Value="XL">XL

<P>Cosa vuoi fare ?
<P><SELECT Name="would-like" size="3" multiple>
<OPTION>Ordina il prodotto</OPTION>
<OPTION>Fai una domanda</OPTION>
<OPTION>Richiesta catalogo</OPTION>
</SELECT>

<P>Invio con il corriere ? <INPUT type = "CHECKBOX" name = "corriere"  value = "si"
checked>

<P>Note<BR>
<TEXTAREA Name="note" rows="4" cols="20"></TEXTAREA>

<P><INPUT Type="submit" Value="Invia">

<INPUT Type="reset" Value="Annulla">

</FORM>
 </BODY>
</HTML>
```

Cognome e Nome :

Codice Articolo :

Taglia della maglia? S M L XL

Cosa vuoi fare ?

Ordina il prodotto
Fai una domanda
Richiesta catalogo

Invio con il corriere ?

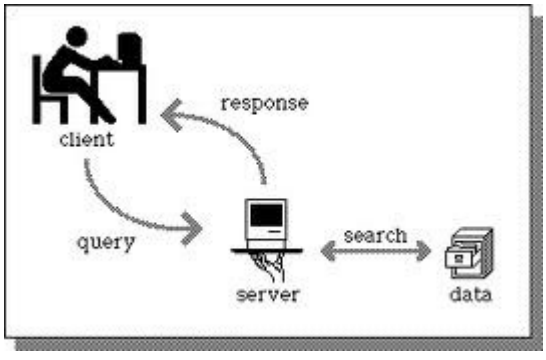
Note

Invia Annulla

IL PHP

Il linguaggio HTM, illustrato in precedenza , impone limitazioni alle applicazioni che si possono realizzare: ad esempio non è possibile realizzare un sito web che acceda ad un database. Questo tipo di linguaggio svolge una sua funzione dal lato del client.
Un' architettura web è composta da due parti che interagiscono fra loro:

- Il lato client (**client-side**);
- Il lato server (server-**side**).



La pagina web, costruita con i tag dell'HTML, in forma di file di testo risiede sul disco fisso del server in una particolare cartella. Quando l'utende nel browser scrive l'indirizzo di una pagina HTML (o di un' altra risorsa), la richiesta viene inviata al server, che recupera la pagina sul suo disco e la restituisce al client senza alcuna modifica. La pagina arrivata al client viene interpretata dal browser e quindi mostrata all'utente.
Per poter interagire con un database che si trova su un server remoto, si devono usare altre tecnologie che effettuano determinate operazioni dal lato del server (server-side).
I due strumenti software che utilizzeremo per la programmazione server-side sono:

- Il linguaggio **PHP**;
- Il DBMS (database management system) **MySql**.

Il PHP è un linguaggio molto simile al C /C++ . Viene interpretato dal server web e consente di *costruire pagine web dinamiche*. Normalmente i file PHP contengono parti scritte in HTML e altre parti scritte in PHP contenute tra speciali Tag. Quando l'interprete PHP inizia ad esaminare un file, visualizzerà il contenuto del file sino a quando non incontra uno dei tag speciali indicanti l'inizio del codice da interpretare come istruzioni PHP. A questo punto il parser eseguirà tutto il codice trovato sino a quando non incontrerà i tag di chiusura, che indicano al parser di tornare alla modalità di visualizzazione: tutto ciò che si trova all'esterno dei tag PHP sarà lasciato
inalterato, mentre tutto ciò che si trova all'interno sarà eseguito come codice PHP.

```
<HTML>
<HEAD>
<TITLE>pagina di prova<TITLE>
<HEAD>
<BODY>
.
.
<?php
  echo "Questa è una costante stringa";
  echo "le istruzioni terminano con il punto e virgola";
  /*
        Come nel C /* e */ indicano inizio e fine di un  commento su più righe
  */
  //    Come nel C++  indica il commento su un'unica  riga

?>
.
.
</BODY>
</HTML>
```

Il tag di chiusura di un blocco ?> include il carattere di 'a capo' immediatamente seguente, se presente. Inoltre, il
tag di chiusura viene considerato automaticamente come punto e virgola; pertanto non occorre inserire il
punto e virgola alla fine dell'ultima riga del blocco php.
Il PHP permette strutture tipo le seguenti:

```
<?php
if ($expression) {
   ?>
        <H2>Questa è vera.</H2>
   <?php
} else {
   ?>
        <H2>Questa è falsa.</H2>
   <?php
}
?>
```

Questo esempio agisce come atteso, poiché il PHP rileva il tag di chiusura ?>, e da questo punto, inizia a dare in output tutto ciò che trova fino a quando non rileva un'altro tag di apertura. Certamente l'esempio dato è macchinoso, ma per l'output di grossi blocchi di testo, l'uscire dalla modalità di parsing PHP, è generalmente più efficiente piuttosto che inviare il testo tramite ripetute funzioni echo() o print().

Tipi di dati

PHP supporta otto tipi primitivi:
Quattro scalari: ***boolean, integer, float, string***
Due tipi composti: ***array, object***
Due tipi speciali: ***resurce, NULL***
Le variabili vengono dichiarate implicitamente, cioè il tipo viene deciso da PHP in base al contesto in cui la variabile è usata.

```
<?php
  $booleano = TRUE;// è un boolean
  $stringa = "Ciao"; // è una string
  $int = 12;              // è un integer
?>
```

Si può forzare la conversione di una variabile in un determinato tipo con l'operatore di cast (come in C o Java) o con la funzione ***settype();*** da notare che la stessa variabile può assumere valori diversi in certe situazioni, in base al tipo che è in quel contesto.

Boolean

Può assumere i due valori TRUE o FALSE. Per introdurre una variabile booleana basta assegnarle il valore TRUE o FALSE (maiuscole o minuscole)

```
<?php
      $test = True; // assegna il valore TRUE alla variabile $test
?>
```

Per convertire esplicitamente un valore in boolean si usano gli operatori di cast (bool) o (boolean), i seguenti valori sono considerati FALSE:

- il boolean FALSE
- l'integer 0 (zero)
- il float 0.0 (zero)
- la stringa vuota e la stringa "0"
- un array con zero elementi
- un oggetto con zero variabili membro
- NULL

Tutti gli altri valori sono considerati TRUE

```
<?php
     echo gettype((bool) ""); // bool(false)
     echo gettype((bool) 1); // bool(true)
     echo gettype((bool) -2); // bool(true)
     echo gettype((bool) "foo"); // bool(true)
     echo gettype((bool) 2.3e5); // bool(true)
     echo gettype((bool) array(12)); // bool(true)
     echo gettype((bool) array()); // bool(false)
?>
```

Il tipo integer

Un integer è un numero intero di lunghezza dipendente dal sistema operativo, in genere 32 bit con segno, può essere specificato in base 10, 16, 8 ed eventualmente preceduto dal segno.

```php
<?php
  $a = 1234; # numero decimale
  $a = -123; # numero negativo
  $a = 0123; # numero ottale (equivalente a 83 decimale)
  $a = 0x1A;  # numero esadecimale (equivalente a 26 decimale)
?>
```

- Se si superano i limiti del tipo intero (**overflow**) questo verrà interpretato come float;
- Non esiste in PHP un operatore di divisione intera, il risultato è sempre float, l'operatore di cast (int) lo tronca, per arrotondare al valore più vicino si usa la funzione **round().**

```php
<?php
  var_dump(25/7); // float(3.5714285714286)
  var_dump((int) (25/7)); // int(3)
  var_dump(round(25/7)); // float(4)
?>
```

La conversione ad intero avviene automaticamente se un operatore, una funzione o una struttura di controllo richiede un argomento intero. Da un boolean FALSE ritorna 0 e TRUE ritorna 1, le stringhe vengono interpretate come interi nei casi che vedremo successivamente, per altri tipi occorre cautela.

Il tipo float

Sono specificati usando i seguenti tipo di sintassi:

```php
<?php
  $a = .234;
  $b = .2e3;
  $c = 7E-10;
?>
```

La dimensione di float dipende dal sistema operativo, normalmente viene usato il formato IEEE a 64 bit che fornisce all'incirca un valore massimo di 1.8e308 con una precisione di circa 14 cifre decimali. Dato che numeri decimali come 0.1 o 0.7 o frazioni come 1/3 non possono essere convertiti esattamente nella rappresentazione float binaria senza una piccola perdita di precisione non confrontate mai l'uguaglianza di due numeri float.

Il tipo string

Una stringa è una serie di caratteri: in PHP la rappresentazione di un carattere è a 8 bit, ci sono quindi esattamente 256 caratteri diversi. Questo implica che non c'è supporto nativo per i codici Unicode, qualche supporto è fornito dalle funzioni **utf8_encode() e utf8_decode().**
La lunghezza di una stringa è indefinita.
Una stringa costante può essere specificata in tre modi:

1. Tra apici (')

Se si vuole inserire un apice nella stringa dovete farlo precedere da backslash (\') , il carattere backslash può essere inserito raddoppiandolo(\\). In questo formato non possono essere inseriti caratteri speciali come \n o \t.

2. Tra virgolette (")

La caratteristica più importante delle stringhe racchiuse tra virgolette è che ***le variabili vengono espanse, cioè sostituite con il loro valore.***
Al suo interno si possono usare i caratteri speciali del linguaggio C, in particolare:

\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\\	backslash
\$	dollar sign
\"	double-quote

\[0-7]	codice ottale del carattere (da 1 a 3 cifre)
\x[0-9A-Fa-f]	codice esadecimale (1 o 2 cifre)

3. Heredoc

In caso la stringa è molto lunga e la si vuole scrivere su più linee, in questo caso la stringa va inserita tra
<<<identificatore
…….
…….
identificatore;
da notare che l'identificatore di chiusura deve necessariamente iniziare nella prima colonna di una riga ed essere identico a quello di apertura. All'interno si possono usare caratteri speciali e le variabili vengono espanse come con le virgolette.

```
<?php
$nome='Luciano';
$str = <<<STRINGA
Esempio di stringa
che si espande su più righe
usando la sintassi heredoc
Posso inserire la variabile
$nome che verrà sostituita
dalla stringa Luciano
STRINGA;
?>
```

Sappiamo che la stringa è un vettore di caratteri quindi il primo carattere ha indice 0, l'indice in PHP 4 va racchiuso tra parentesi graffe {};

```
<?php
 $str = 'Stringa di prova';
 $first = $str{0};              // Prende il primo carattere di una stringa
 $last = $str{strlen($str)-1}; // Prende l'ultimo carattere di una stringa
?>
```

Conversione in stringa

Si può convertire un valore in stringa usando il cast (string) o la funzione **strval().**
La conversione in stringa avviene automaticamente con le funzioni **echo() o print()** o quando un valore è confrontato con una stringa. TRUE viene convertito in '1' e FALSE nella stringa vuota (in questo modo è possibile la conversione inversa). Un numero intero o float è convertito nella sua rappresentazione.
Il contenuto di un array non è convertito automaticamente in stringa, si ottiene la stringa 'Array' ma possono esserlo i suoi elementi, in modo simile si comportano gli Object.
NULL è sempre convertito nella stringa vuota.

Conversione di stringhe in numeri

Se una stringa contiene all'inizio un valore numerico in un formato valido viene convertita in numero.

```
<?php
$f = 1 + "10.5";              // $f è float (11.5)
$f = 1 + "-1.3e3";            // $f è float(-1299)
$f = 1 + "bob-1.3e3";         // $f è integer (1)
$f = 1 + "bob3";              // $f è integer (1)
$f = 1 + "10 Small Pigs";           // $f è integer (11)
$f = 4 + "10.2 Little Piggies"; // $f è float (14.2)
$f = "10.0 pigs " + 1;        // $f è float (11)
$f = "10.0 pigs " + 1.0;      // $f è float (11)
?>
```

Il tipo array

Un array in PHP è una mappa ordinata, cioè un tipo che associa una chiave a un valore. Viene creato con il seguente costrutto sintattico:

array([key =>] value,...);

dove key può essere un intero o una stringa e value un qualsiasi valore, compreso un altro array.

```
<?php
 $arr = array( "primo" => "bar", 12 => true );
 echo $arr["primo"];   // avremo  bar
 echo $arr[12];                  // avremo  1
?>
```

- se non si specifica la chiave viene preso il massimo indice intero aumentato di 1;
- se specificate una chiave che esiste già il valore è sovrascritto.

```
<?php
// Questi due array sono gli stessi
array( 5 => 43, 32, 56, "b" => 12 );
array( 5 => 43, 6 => 32, 7 => 56, "b" => 12 );
?>
```

- se il valore massimo è negativo il nuovo valore è 0 .
- un array può essere modificato scrivendo in esso valori esplicitamente con la sintassi:

$arr[key] = valore;

- se l'array non esiste viene creato
- se key non è specificata viene preso l'ultimo indice intero aumentato di 1
- per rimuovere un elemento si usa la funzione **unset().**

```
<?php
 $arr = array(5 => 1, 12 => 2);
 $arr[] = 56;     // Come $arr[13] = 56;
 $arr["x"] = 42; // aggiunge un nuovo elemento con chiave x
 unset($arr[5]); // rimuove l'elemento
 unset($arr)      ;// Cancella l'intero array
?>
```

Per ciascuno dei tipi: integer, float, string, boolean e resource, se convertite il valore in array ottenete un array con un solo elemento con indice 0.

Gli array sono ordinati, l'ordine può essere cambiato con svariate funzioni di ordinamento, la funzione **count()** ne conta gli elementi.

Il tipo object

Il PHP consente di creare oggetti e di usarli in modo simile a C++ e Java, per esempio:

```
<?php
class oggetto              //definizione dalla classe
{
   function esempio ( )
   {       echo "Ho eseguito oggetto";    }
}
$bar = new oggetto;    //creo un  oggetto
$bar -> esempio( );     //  eseguo metodo
?>
```

Il tipo resource

È una variabile speciale che contiene un riferimento a una risorsa esterna creata e usata da

funzioni speciali, per esempio un puntatore a un file aperto, una connessione a un database e così via. Nessun altro valore può essere convertito in una risorsa. PHP libera automaticamente le risorse non usate, ad eccezione delle connessioni persistenti a un database.

Il tipo NULL
Il valore speciale NULL rappresenta una variabile senza alcun valore ed è l'unico valore possibile per il tipo NULL introdotta in PHP 4.
Una variabile è considerata NULL se:

- gli è stata assegnata la costante NULL;
- non gli è stato ancora assegnato un valore;
- su di essa è stata applicata la funzione unset().

Variabili
Le variabili in PHP sono rappresentate dal simbolo di dollaro seguito dal nome della variabile, il nome è case-sensitive. Le variabili possono essere assegnate sia *per valore* che **per riferimento**, ciò significa che la nuova variabile è un sinonimo (alias) di quella originale: *modificandone una modifico anche l'altra*. In questo modo si guadagna in velocità e in spazio in memoria, in particolare con variabili molto grandi. L'assegnazione per riferimento avviene ponendo prima della variabile il simbolo &.

```
<?php
  $a = 'pippo';
  $b = &$a;              //  a viene assegnata per riferimento ad  a
  $b = "Il mio nome è $b";
  echo $a;               // viene scritto "Il mio nome è pippo"
?>
```

Da notare che solo le variabili con nome possono essere assegnate per riferimento, $a=&(3+2); non è valida.
Non è nemmeno concesso far riferimento al valore di ritorno di una funzione.

Variabili predefinite
PHP fornisce un gran numero di variabili predefinite, molte di queste tuttavia dipendono dal particolare server in esecuzione, dalla sua versione e dalla sua configurazione.
Dalla versione 4.1.0 PHP fornisce inoltre un insieme di array predefiniti contenenti variabili del server web, questi array sono sempre disponibili e sono noti come autoglobals o superglobals.

$GLOBALS	Contiene un riferimento a ciascuna variabile globale correntemente disponibile, le chiavi sono i nomi delle variabili.
$_SERVER	Variabili settate dal server web o direttamente correlate all'ambiente di esecuzione dello script corrente
$_GET	Variabili fornite dal metodo GET HTTP
$_POST	Variabili fornite dal metodo POST HTTP
$_COOKIE	Variabili fornite dai cookie HTTP
$_FILES	Variabili fornite all'upload di file
$_ENV	Variabili fornite dall'ambiente
$_REQUEST	Variabili fornite dai metodi di input GET, POST e COOKIE
$_SESSION	Variabili di sessione correntemente registrate

Variabili esterne a PHP
Vengono usate per leggere il contenuto di una form con il metodo GET e POST, per esempio una form HTML potrebbe essere la seguente:

```
<form action="ricevi.php" method = "POST">
  Nome: <input type="text" name="username"><br>
  Email: <input type="text" name="email"><br>
  <input type="submit" name="submit" value="Invia">
</form>
```

Ci sono vari modi con cui il file *ricevi.php* può ottenere i dati, eccone alcuni:

```
<?php
  // Disponibile da PHP 4.1.0
  print $_POST['username'];
  print $_REQUEST['username'];
  import_request_variables('p', 'p_');
  print $p_username;
?>
```

L'accesso ai dati di una form che usa il metodo GET è simile, usando le variabili predefinite.
Nota: Se nel nome di una variabile esterna viene usato il punto PHP lo converte in un trattino basso (underscore)

Nomi di variabili con IMAGE SUBMIT

In una form è possibile usare un'immagine al posto del pulsante submit con la seguente sintassi:

<input type="image" src="image.gif" name="sub">

quando l'utente "clicca" con il mouse in un punto dell'immagine, il form viene trasmessa con la variabile di nome sub e due variabili addizionali: sub_x e sub_y che contengono le coordinate del punto all'interno dell'immagine.

HTTP Cookie

PHP supporta in modo trasparente i cookie come definiti dalle specifiche Netscape. I cookie sono creati con la funzione **setcookie(),** dato che i cookie fanno parte dell'header HTTP la funzione deve essere chiamata prima di inviare qualsiasi output al browser. I dati dei cookie sono disponibili negli array di dati $_COOKIE, $HTTP_COOKIE_VARS e $_REQUEST.

Variabili static

Come nel linguaggio C una variabile static non perde il suo valore quando l'esecuzione di una funzione termina, possono essere usate per realizzare contatori, per esempio:

```
<?php
  function prova( )
    {
                  static $a = 0;
                   echo $a;
                   $a++;
  }
?>
```

In questo caso il valore di $a viene incrementato e scritto ogni volta che la funzione viene chiamata, l'assegnazione $a=0; è eseguita solo la prima volta che la funzione è chiamata. Ad una variabile statica non può essere assegnata una espressione, per esempio static $a=1+1; non è un'espressione valida.

Variabili variabili

Talvolta è conveniente avere nomi di variabili che possono essi stessi cambiare, l'assegnazione viene fatta con un doppio simbolo di dollaro, per esempio:

```
<?php
    $$a = "ciao";
?>
```
crea una nuova variabile con nome $ciao a cui ci si può riferire con ${$a}.

Costanti simboliche
Attraverso un identificatore indichiamo una costante che come si può intuire non può cambiare durante l'esecuzione dello script. Una costante è "case-sensitive" per default. È convenzione comune che i nomi di costante siano sempre maiuscoli.
Le costanti possono contenere solo dati di tipo scalare (boolean, integer, float e string).
Come le superglobals, una costante simbolica è sempre globale cioè si può accedere alle costanti da qualsiasi punto dello script senza tenere conto della visibilità.
Si può anche utilizzare la funzione **constant(),** per leggere il valore di una costante, nel caso in cui se ne ottenga dinamicamente il nome. Si utilizzi **get_defined_constants()** per ottenere una lista delle costanti definite.
Di seguito sono riportate le principali differenze rispetto alle variabili:

- Le costanti non iniziano con il segno del dollaro ($);
- Le costanti possono essere definite **solo** con la funzione **define()** e non tramite assegnazione;
- Le costanti possono essere definite e utilizzate ovunque senza seguire le regole di visibilità;
- Una volta impostate, le costanti non possono essere ridefinite né annullate;
- Le costanti possono essere solo valori scalari;

```
<?php
    define("PIPPO", "Ciao mondo.");
    echo PIPPO   ;        // stampa "Ciao mondo."
?>
```

Costanti predefinite
Il PHP mette a disposizione ad ogni script diverse costanti predefinite. Alcune di queste, tuttavia, sono create dai vari moduli, e, pertanto, saranno disponibili solo quando questi moduli sono caricati, sia dinamicamente sia staticamente. Esistono quattro costanti magiche il cui valore cambia in base al contesto in cui sono utilizzate. Ad esempio, il valore di _LINE_ dipende da quale linea si trova nel momento in cui è richiamata. Queste costanti speciali sono 'case-insensitive' e sono:

`_LINE_`	Il numero di linea corrente.
`_FILE_`	Il nome e percorso assoluto del file.
`_FUNCTION_`	Nome della funzione.
`_CLASS_`	Nome della classe.
`_METHOD_`	Nome del metodo della classe

Espressioni
Il modo più semplice ma accurato per definire un'espressione è "qualsiasi cosa che ha un valore". Una particolarità di PHP è di essere un linguaggio orientato alle espressioni, quasi tutto è un'espressione. Consideriamo per esempio una semplice assegnazione:
$a = 5;
Si vede facilmente che ci sono due valori coinvolti, la costante 5 e la variabile $a, ma in realtà c'è un valore in più coinvolto: l'assegnazione stessa, cioè la stringa '$a=5' che assume anch'essa il valore 5.Per esempio queste tre assegnazioni sono equivalenti, per quanto riguarda i valori assunti dalle variabili $a e $b:

```
$b=($a=5);
$a=5; $b=5;
$b=$a=5;
```

Si ricorda che le assegnazioni sono valutate da destra a sinistra.
In genere le espressioni contengono operatori, per esempio

- operatori di relazione ==, !=, >,... che forniscono come risultato TRUE o FALSE
- operatori aritmetici (+ , -, *)
- operatore di assegnazione che può essere combinato con l'assegnazione come in C (`$a *=5;`)
- :? l'operatore condizionale ternario

```
<?php
$primo ? $secondo : $terzo
?>
```

se il valore della prima espressione è TRUE (diverso da 0) viene valutata la seconda espressione altrimenti viene valutata la terza.

Operatori

Gli operatori sono simili a quelli del linguaggio C++, ci limiteremo a descrivere quelli che non esistono in C++ o quelli che funzionano in modo diverso.
Richiamiamo due importanti proprietà degli operatori: la **precedenza** e l'**associatività**.
La precedenza stabilisce l'ordine in cui vengono applicati gli operatori presenti nell'espressione.
L'**associatività** invece si riferisce invece all'ordine in cui vengono eseguiti operatori con la medesima precedenza, per esempio gli operatori aritmetici di pari precedenza, che hanno associatività sinistra vengono eseguiti da sinistra verso destra.

Associatività	**Operatori quelli con precedenza più bassa sono elencati per primi**
sinistra	,
sinistra	or
sinistra	xor
sinistra	and
destra	Print
sinistra	`= += -= *= /= .= %= &= |= ^= ~= <<= >>=`
sinistra	`? :`
sinistra	`||`
sinistra	&&
sinistra	`|`
sinistra	^
sinistra	&
non associativi	`== != === !==`
non associativi	`< <= > >=`
sinistra	`<< >>`
sinistra	`+ - .`
sinistra	`* / %`
destra	`! ~ ++ -- (int) (float) (string) (array) (object) @`
destra	[
non associativi	new

Operatori di controllo errori (@)

Se @ viene anteposto ad una espressione in PHP, qualunque messaggio di errore che potesse essere generato da quella espressione sarà ignorato.
Se la caratteristica **track_errors** è abilitata, qualsiasi messaggio di errore generato

dall'espressione sarà salvato nella variabile globale **$php_errormsg**. Questa variabile sarà sovrascritta ad ogni errore, quindi va controllata subito se necessario.

```
<?php
    /*  Errore di file intenzionale  */
    $my_file = @file ( 'file_inesistente ')  or
    die ( "Apertura del file fallita: l'errore è '$php_errormsg'" );
    // questo funziona per qualsiasi espressione, non solo funzioni:
    $value = @$cache[$key];
    // non verrà generata una notifica se l'indice $key non esiste.
?>
```

Operatori di stringa

Sono due

'.' concatenazione

'.=' assegnazione concatenata (aggiunge alla fine dell'argomento di destra l'argomento di sinistra)

```
    $a = "Ciao ";
    $b = $a . "Mondo!"; // ora $b contiene "Ciao Mondo!"
    $a = "Ciao ";
    $a .= "Mondo!";     // ora $a contiene "Ciao Mondo!"
```

Strutture di controllo

Qualsiasi script PHP è costituito da una serie di istruzioni. Una istruzione può essere un'assegnazione, una chiamata di funzione, un loop, una istruzione condizionale che non fa nulla (istruzione vuota). Le istruzioni terminano con un punto e virgola. Inoltre, le istruzioni si possono raggruppare in blocchi di istruzioni racchiudendole tra parentesi graffe. Un gruppo di istruzioni è, a sua volta, un'istruzione. Anche in questo caso ci limiteremo a descrivere le differenze rispetto al linguaggio C++;

Sintassi alternativa per le strutture di controllo

PHP offre una sintassi alternativa per alcune delle sue strutture di controllo; vale a dire, **if, while,for, foreach e switch**. Fondamentalmente la sintassi alternativa consiste nel sostituire la prima parentesi graffa con il carattere "due punti" (:) e la seconda parentesi graffa con **endif, endwhile, endfor, endforeach, endswitch**, rispettivamente.

```
<?php
        if ($a == 5):
 ?>
      a è uguale a 5
<?php
 endif;
?>
```

Nell'esempio precedente, il blocco HTML "a è uguale a 5" è incluso nel ramo if scritto utilizzando la sintassi alternativa. Il blocco HTML verrà visualizzato solamente se $a è uguale a 5. Questa sintassi è utile quando sivogliono produrre in modo condizionale due spezzoni di pagina HTML (o due intere pagine HTML).

foreach

Come per il Perl il PHP 4 (non PHP 3) permette l'uso della struttura di controllo foreach

foreach (array_expression as $value) istruzione;
foreach (array_expression as $key => $value) istruzione;

La prima attraversa l'array dato da array_expression. Ad ogni ciclo, si assegna il valore

dell'elemento corrente a $value e il puntatore interno avanza di una posizione (in modo tale che al ciclo successivo l'elemento corrente sarà il successivo elemento dell'array).
La seconda esegue lo stesso ciclo con la differenza che il valore dell'indice corrente viene assegnato ad ogni ciclo, alla variabile $key.
Nota: All'inizio dell'esecuzione di un ciclo foreach il puntatore interno viene automaticamente posizionato nella prima posizione. Questo significa che non è necessario utilizzare la funzione **reset()** prima di un ciclo foreach.
Nota: È importante notare che foreach opera su una copia dell'array, non sull'array stesso, pertanto il puntatore dell'array originale non viene modificato come accade utilizzando la funzione each() e le modifiche agli elementi dell'array non appaiono nell'array originale.

break
break termina l'esecuzione di una struttura for, foreach, while, do..while o switch.
break accetta un argomento opzionale che definisce, nel caso di cicli annidati, il livello del ciclo che è da interrompere.

```
/* Uso dell'argomento opzionale. */
$i = 0;
while (++$i) {
   switch ($i) {
   case 5:
      echo "At 5<br>\n";
      break 1; /* Interrompe solo switch. */
   case 10:
      echo "At 10; quitting<br>\n";
      break 2; /* Interrompe switch e while. */
   default:
      break;
   }
}
```

continue
Si utilizza per interrompere l'esecuzione del ciclo corrente e continuare con l'esecuzione all'inizio del ciclo successivo; continue accetta un argomento numerico opzionale che definisce, nel caso di cicli annidati, il numero di cicli da interrompere e da cui iniziare l'esecuzione dell'iterazione successiva.
declare
Il costrutto declare si usa per definire direttive di esecuzione per blocchi di istruzioni. La sintassi è simile alla sintassi di altre strutture di controllo:
declare (direttiva) istruzione
La sezione direttiva permette di impostare il comportamento del blocco declare . Attualmente è riconosciuta una sola direttiva: la direttiva ticks.
tick
Un tick è un evento che si verifica per ogni N istruzioni di basso livello eseguite dal parser all'interno del blocco declare. Il valore per N viene specificato usando ticks = N all'interno della sezione direttiva del blocco declare.
L'evento (o gli eventi) che si verifica su ogni tick è specificato usando register_tick_function().
Vedere l'esempio più in basso per ulteriori dettagli. Notare che può verificarsi più di un evento per ogni tick.

```
<?php
// Una funzione che registra il tempo quando viene chiamata
function profile ($dump = FALSE)
{
    static $profile;
    // Restituisce il tempo in $profile, successivamente lo cancella
    if ($dump) {
        $temp = $profile;
        unset ($profile);
        return ($temp);
    }
    $profile[] = microtime ();
}
// Imposta un tick handler
register_tick_function("profile");
// Inizializza la funzione prima del blocco declare
profile ();
// Esegue un blocco di codice, attraverso un tick ogni seconda istruzione
declare (ticks = 2) {
    for ($x = 1; $x < 50; ++$x) {
        echo similar_text (md5($x), md5($x*$x)), "<br />";
    }
}
// Mostra i dati immagazzinati nel profilo
print_r (profile (TRUE));
?>
```

L'esempio descrive il codice PHP all'interno del blocco declare, registrando il tempo in cui è stata eseguita ogni seconda istruzione di basso livello. Questa informazione può poi essere usata per trovare le aree lente all'interno di particolari segmenti di codice. Questo processo può essere ottimizzato usando altri metodi: usando i tick è più conveniente e facile da implementare. I tick sono ben adeguati per il debugging, l'implementazione di semplici multitasking, backgrounded I/O e molti altri compiti.

require()

L'istruzione require() inserisce e valuta il file specificato in parantesi o ‘o assegnato a variabile. require() e include() sono identiche eccetto per come esse trattano gli errori.

- include() produce un Warning
- require() restituisce un Fatal Error.

```
<?php
  require 'prepend.php';
   require $qualche_file;
  require ('qualche_file.txt');
?>
```

include()

L'istruzione include() include e valuta il file specificato.

La documentazione seguente si applica anche a require(). I due costrutti sono identici in ogni aspetto eccetto per come essi trattano gli errori.

Quando un file viene incluso, il codice che esso contiene eredita lo scope delle variabili della riga in cui si verifica l'inclusione. Qualsiasi variabile disponibile in quella riga nella chiamata al file sarà disponibile all'interno del file chiamato, da quel punto in avanti.

```
vars.php
<?php
  $colore = 'verde';
  $frutto  =  'mela';
?>
test.php
<?php
 echo "Una $frutto $colore";   //    Output: Una
 include 'vars.php';
 echo "Una $frutto $colore";   //    Output: Una mela verde
?>
```

Quando un file viene incluso, il parsing esce dalla modalità PHP e entra in modalità HTML all'inizio del file incluso, e riprende alla fine. Per questa ragione, qualunque codice all'interno del file incluso che dovrebbe essere eseguito come codice PHP deve essere incluso all'interno dei tag PHP validi di apertura e chiusura. Se "**URL fopen wrappers**" nel PHP sono abilitati (come nella configurazione di default), potete specificare il file da includere usando un URL (via http) invece che un percorso locale. Poichè include() e require() se si trovano in un blocco condizionale, dovino essere inseriti all'interno di blocchi di istruzioni.

```
<?php

if ($condizione)  // NON CORRETTO
    include $primo;
else
    include $secondo;

if ($condizione) // CORRETTO
 {
    include $primo;
 } else
 {
    include $secondo;
 }
?>
```

require_once()

Le istruzioni **require_once(), include_once()**
hanno lo stesso comportamento di require() e include() con la differenza che se il codice di un file è stato già incluso, esso non sarà incluso nuovamente.

Funzioni

Una funzione può essere definita usando la seguente sintassi:

function *nome_funzione* (*$arg_1, $arg_2, ..., $arg_n*)

```
 {
    echo "Funzione di esempio.\n";
    return $retval;
 }
```

All'interno di una funzione può apparire qualunque codice PHP valido, persino altre funzioni e definizioni di classe.
L'informazione può essere passata alle funzioni tramite la lista degli argomenti, che sono liste di variabili e/o costanti delimitati dalla virgola.
PHP supporta il passaggio di argomenti per valore (comportamento di default), il passaggio

per riferimento, e i valori di default degli argomenti. Le liste di argomenti di lunghezza variabile sono supportate solo in PHP 4 e successivi: vedere Liste di argomenti a lunghezza variabile e i riferimenti alle funzioni func_num_args(), func_get_arg(), e func_get_args() per maggiori informazioni.
Se si vuole che la funzioni possa modificare i parametri il passaggio non può avvenire valore ma per riferimento. Per indicare questo tipo di passaggio bisogna anteporre una **&** al nome dell'argomento nella definizione della funzione:

```
function  prova (  &$string  )
 {
     $string .= 'Giuseppe ';
 }
$str = 'Rossi  ';
prova (  $str  );
echo $str;         // l'output sarà 'Rossi  Giuseppe'
```

Come in C++ una funzione può avere valori predefiniti o di default:

```
function fare_il_caffe (  $tipo = "cappuccino" )
{
   return "Sto facendo una tazza di $tipo.\n";
}
```

Se la funzione viene chiamata echo fare_il_caffe () come parametro attuale viene preso cappuccino;
Il valore predefinito deve essere un'espressione costante, non (per esempio) una variabile o un membro di classe.Quando vengono usati argomenti predefiniti, qualunque argomento predefinito dovrebbe essere a destra degli argomenti non-predefiniti;

Valori restituiti
I valori vengono restituiti usando l'istruzione opzionale return. Può essere restituito qualsiasi tipo, incluse liste ed oggetti. Ciò provoca l'interruzione dell'esecuzione della funzione immediatamente e la restituzione del controllo alla linea successiva a quella da cui è stata chiamata.

```
function quadrato ($num)
{
   return $num * $num;
}
echo quadrato (4);        //   L'output è '16'
```

.
Non possono essere restituiti valori multipli ma array si.

```
function prova  ()
{
    return array (0, 1, 2);
}

list ($zero, $uno, $due)  =  prova ();
```

Per restituire un riferimento da una funzione, è necessario usare l'operatore di passaggio per riferimento & nella dichiarazione di funzione e quando viene assegnato il valore restituito ad una variabile:

```
function &restituisce_riferimento()
{
   return $un_riferimento;
}
$nuovo_riferimento =& restituisce_riferimento();
```

Funzioni variabili

Se il nome di una variabile ha le parentesi è seguita da parentesi, *PHP cercherà una funzione con lo stesso nome del valore della variabile*, e cercherà di eseguirla. Tra le altre cose, ciò può essere usato per implementare delle callbacks, tabelle di funzioni e così via.

Le funzioni variabili non funzionano con costrutti di linguaggio come echo(), unset(), isset(), empty() e include(). Il costrutto print() è un'eccezione e funzionerà.

```
<?php
    function prima( )
      {
        echo "In prima()<br>\n";
      }
    function seconda  ( $arg = '  ' )
      {
        echo "In seconda(); l'argomento era '$arg'.<br>\n";
      }
    $func = 'prima';
    $func();
    $func = 'seconda';
    $func('test');
?>
```

Classi e Oggetti

Una classe è una collezione di variabili e funzioni che utilizzano queste variabili. Una classe si definisce usando la seguente sintassi:

```php
<?php
 class Cart                // Articoli nel carrello
   {
         var $items;
       //          Aggiunge $num articoli di $artnr nel carrello
       function aggiungi (  $artnr, $num  )
         {
             $this->items[$artnr] += $num;
         }
       //          Prende $num articoli di $artnr e li rimuove dal carrello
       function rimuovi (  $artnr, $num  )
         {
                    if ( $this->items[$artnr] > $num )
                    {
                       $this -> items[$artnr] -= $num;
                       return true;
                     } else
                     {
                       return false;
                      }
         }
    }
?>
```

Il codice definisce una classe chiamata Cart composta da un array associativo che archivia gli articoli nel carrello e due funzioni per aggiungere e rimuovere gli articoli dal carrello stesso.
In PHP 4, sono permesse inizializzazioni di variabili con valori costanti solamente grazie all'uso di var. Per inizializzare variabili con valori non-costanti, bisogna creare una funzione d'inizializzazione (costruttore) che è chiamata automaticamente alla dichiarazione di un oggetto istanza della classe.

```php
<?php
 class Cart
   {
        var $todays_date;
        var $name;
        var $owner;
        var $items;
        function Cart()
          {
              $this -> todays_date = date("Y-m-d");
              $this -> name = $GLOBALS['firstname'];
           }
   }
?>
```

Le classi sono tipi del linguaggio, e sono modelli per variabili reali. Per creare un oggetto si usa l'operatore new.

```php
<?php
        $cart = new  Cart;
        $cart -> aggiungi( "10", 1 );
        $another_cart  =  new Cart;
        $another_cart -> aggiungi ( "0815", 3 );
?>
```

Vengono generati gli oggetti $cart e $another_cart, dalla classe Cart. La funzione aggiungi() dell'oggetto $cart è chiamata per aggiungere una ricorrenza dell'articolo numero 10 a $cart. A $another_cart sono aggiunte 3 pezzi dell'articolo numero 0815.

Classe derivata (o estesa)

Spesso si ha bisogno di avere classi con variabili e funzioni simili ad altre classi. É buona norma definire una classe in modo generico, sia per poterla riutilizzare spesso, sia per poterla adattare a scopi specifici. Per facilitare questa operazione, è possibile generare classi per estensione di altre classi. Una **classe estesa o derivata** ha tutte le variabili e le funzioni della classe di base più tutto ciò che viene aggiunto dall'estensione.
Non è possibile che una sottoclasse, ridefinisca variabili e funzioni di una classe madre. Una classe estesa dipende sempre da una singola classe di base: l'eredità multipla non è supportata. Le classi si estendono usando la parola chiave 'extends'.

```
<?php
    class Named_Cart extends Cart
        {
           var   $owner;
           function set_owner ($name)
            {
                  $this->owner = $name;
             }
        }
?>
```

Qui viene definita una classe Named_Cart che ha tutte le funzioni e variabili di Cart più la variabile $owner e la funzione set_owner(). Viene creato un carrello con nome con il metodo usato in precedenza, in più la classe estesa permette di settare o leggere il nome del carrello.
Si possono usare variabili e funzioni sia di Cart che della sua estensione:

```
<?php
       $ncart = new Named_Cart;      // Crea un carrello con nome
       $ncart -> set_owner( "kris" );  // Assegna il nome al carrello
       print $ncart -> owner;          // stampa il nome del proprietario
       $ncart -> aggiungi ( "10", 1 );  // (funzionalità ereditata da Cart)
?>
```

Costruttori

I costruttori sono funzioni che esistono in una classe e che sono chiamate automaticamente quando si crea una nuova istanza di una classe con new. In PHP 3, una funzione si trasforma in un costruttore quando ha lo stesso nome di una classe. In PHP 4, una funzione diventa un costruttore, quando ha lo stesso nome di una classe ed è definita all'interno della classe stessa la differenza è sottile, ma cruciale.

```
<?php
//    funziona in PHP 3 e PHP 4.
  class Auto_Cart  extends  Cart
   {
          function Auto_Cart()
          {
             $this -> aggiungi ( "10", 1 );
          }
   }
?>
```

Questo codice definisce una classe Auto_Cart, che non è altro che Cart più un costruttore che

inizializza il carrello con una occorrenza dell'articolo numero "10" ogni volta che un nuovo Auto_Cart è creato con new. I costruttori possono avere degli argomenti, e gli argomenti possono essere facoltativi, questo li rende molto versatili. *Per poter usare una classe senza specificare parametri, tutti i parametri del costruttore devono essere resi facoltativi con valori di default.*

```
<?php
      // Funziona in PHP 3 and PHP 4.
 class Constructor_Cart extends Cart
  {
       function Constructor_Cart ( $item = "10", $num = 1 )
        {
          $this->aggiungi (  $item, $num  );
        }
  }
  $default_cart  =  new Constructor_Cart;
  $different_cart = new Constructor_Cart("20", 17);
?>
```

serialize()

L'operazione di serializzazione di un oggetto restituisce una stringa che contiene una rappresentazione in caratteri di tutti i valori che possono essere memorizzati in PHP. **unserialize()** può usare questa stringa per ricreare dai valori variabili utilizzabili.

Usando **serialize()** per salvare un oggetto si salveranno tutte le variabili dell'oggetto. Le funzioni dell'oggetto non sono salvate, viene salvato solo il nome della classe.

Per potere usare unserialize() su un oggetto, la classe dell'oggetto deve essere definita. Cioè se avete un oggetto $a istanza della classe A su una pagina di nome page1.php e usate serialize(), otterrete una stringa che si riferisce alla classe A e contiene tutti i valori delle variabili contenute in $a. Se desiderate poter deserializzare l'oggetto in un'altra pagina chiamata page2.php, dovete ricreare $a dalla classe A, la definizione della classe A perciò deve essere presente nella pagina page2.php. Questo può essere fatto per esempio memorizzando la definizione della classe A in un file che viene incluso sia in page1.php che in page2.php.

```
<?php
// ----------------------    classa.inc
   class A
   {
            var $one = 1;
            function show_one()
            {
               echo  $this  ->  one;
            }
   }
// ----------------------    page1.php:
          include("classa.inc");
          $a =  new A;
          $s =  serialize(  $a  ); //        memorizzare $s in qualche posto della page2.
          $fp = fopen(  "store", "w"  );
          fputs (  $fp,  $s  );
          fclose (  $fp  );
// ----------------------    page2.php:
        include("classa.inc"); //      è necessario perché unserialize() funzioni correttamente.
        $s = implode (  "", @file("store")  );
        $a = unserialize ( $s );
        $a -> show_one(); //        ora usiamo la function show_one() dell'oggetto $a.
?>
```

Se state usando le sessioni ed usate session_register() per registrare oggetti, questi oggetti vengono serializzati automaticamente alla fine di ogni pagina PHP e sono deserializzate automaticamente su ogni pagina della sessione. Ciò significa che gli oggetti possono mostrarsi in ogni pagina e che sono parte integrante della sessione.
Si suggerisce vivamente di includere le definizioni delle classi degli oggetti registrati su tutte le pagine, anche se le classi non sono usate su tutte le pagine. Se un oggetto viene deserializzato senza la relativadefinizione della classe, perderà l'associazione ad essa e si trasformerà in un oggetto della classe stdClass senza nessuna funzione disponibile, diventando inutile.Così se nell'esempio qui sopra $a diventasse parte di una sessione e fosse registrato con session_register("a"), dovreste includere un file classa.inc su tutte le pagine in cui è valida la sessione, non soltanto nella page1.php e nella page2.php.

Le funzioni magiche __sleep e __wakeup
serialize() controlla se la vostra classe ha una funzione dal nome magico **__sleep**. In caso affermativo, quella funzione viene eseguita prima di qualsiasi serializzazione. La funzione può pulire l'oggetto e restituire un array con i nomi di tutte le variabili di quell' oggetto che dovrebbero essere serializzate.
Si intende usare __sleep quando chiudendo un collegamento ad un database l'oggetto può avere dati pendenti e l'oggetto ha bisogno di essere ripulito. Inoltre, la funzione è utile se avete oggetti molto grandi che non devono essere salvati completamente.
Per contro, unserialize() controlla per vedere se c'è nella classe una funzione dal nome magico **__wakeup**. Se è presente questa funzione può ricostruire qualunque risorsa che l'oggetto aveva.
L'intento di __wakeup è quello di ristabilire le connessioni ai database che possono esser state persi durante la serializzazione ed effettuare altre mansioni reinizializzazione.

Confronto di oggetti in PHP 4

In PHP 4, gli oggetti sono confrontati semplicemente, cioè: due istanze di oggetto sono uguali se hanno gli stessi attributi e valori, e sono istanze della stessa classe. Questa regola regola è applicata anche nel confronto di due oggetti utilizzando l'operatore di identità (===).

```
<?php
    function bool2str($bool) {
     if ($bool === false) {
     return 'FALSE';
     } else {
     return 'TRUE';
     }
    }
    function compareObjects(&$o1, &$o2) {
     echo 'o1 == o2 : '.bool2str($o1 == $o2)."\n";
     echo 'o1 != o2 : '.bool2str($o1 != $o2)."\n";
     echo 'o1 === o2 : '.bool2str($o1 === $o2)."\n";
     echo 'o1 !== o2 : '.bool2str($o1 !== $o2)."\n";
    }
    class Flag {
     var $flag;
     function Flag($flag=true) {
     $this->flag = $flag;
     }
    }
    class SwitchableFlag extends Flag {
     function turnOn() {
     $this->flag = true;
     }
     function turnOff() {
     $this->flag = false;
     }
    }
    $o = new Flag();
    $p = new Flag(false);
    $q = new Flag();
    $r = new SwitchableFlag();
    echo "Confronto di istanze create con gli stessi parametri\n";
    compareObjects($o, $q);
    echo "\nConfronto di istanze create con parametri diversi\n";
    compareObjects($o, $p);
    echo "\nConfronto di un'istanza della classe genitore con una
    sottoclasse\n";
    compareObjects($o, $r);
?>
```

Si ha: Confronto di istanze create con gli stessi parametri

```
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE
```

Confronto di istanze create con parametri diversi

```
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Confronto di un'istanza della classe genitore con una sottoclasse

```
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Questo è l'output che si ottiene secondo le regole di confronto descritte sopra. Solo le istanze con gli stessi valori per gli attributi e derivanti dalla stessa classe sono considerate uguali ed

identiche.

Funzioni Mysql
Questo esempio mostra come connettersi, eseguire una query, stampare le righe risultanti e disconnettersi dal database MySQL.

```
<?php
/*        Connessione e selezione del database */

 $connessione = mysql_connect( "host", "utente", "password" ) or die (" errore");

   mysql_select_db( "mio_database" )  or die ( "selezione del database non riuscita" );

   $query = "SELECT * FROM mia_tabella";                          //  preparazione di una query
   $risultato = mysql_query ( $query )  or die  ( "Query fallita" ); //  esecuzione della query SQL

/*        Stampa dei risultati in HTML */
   print "<table>\n";
   while ( $linea = mysql_fetch_array ( $risultato, MYSQL_ASSOC ) ) {
            print "\t<tr>\n";
            foreach ($linea as $valore_colonna) {
               print "\t\t<td>$valore_colonna</td>\n";
            }
            print "\t</tr>\n";
        }
   print "</table>\n";

 mysql_free_result ( $risultato );                    //     Liberazione delle risorse del risultato
   mysql_close ( $connessione );                      //     Chiusura della connessione
?>
```

resource mysql_connect ([string server [, string utente [, string password [, bool nuova_connessione [, int client_flags]]]]])
In caso di successo restituisce l'identificativo di una connessione MySQL oppure FALSE in caso di fallimento. Se si fa una seconda chiamata a mysql_connect() con gli stessi argomenti, non viene stabilita nessuna nuova connessione e viene restituito l'identificativo della connessione già aperta. Il parametro nuova_connessione modifica questo comportamento e fa sì che mysql_connect() apra sempre una nuova connessione, anche se i parametri sono gli stessi. Il parametro client_flags può essere combinato con le costanti MYSQL_CLIENT_COMPRESS, MYSQL_CLIENT_IGNORE_SPACE o MYSQL_CLIENT_INTERACTIVE.

bool mysql_select_db (string nome_database [, resource id_connessione])
Imposta il database da attivare sulla connessione specificata o in mancanza di essa, sull'ultima connessione aperta. Se nessuna connessione è aperta, la funzione proverà a stabilire una connessione come se mysql_connect() fosse richiamata senza argomenti ed userà questa. Ogni chiamata successiva a mysql_query() sarà fatta sul database attivo. Vedere anche: mysql_connect(), mysql_pconnect() e mysql_query().

resource mysql_query (string query [, resource id_connessione [, int modo_risultato]])
Invia una query al database attivo. Se identificativo_connessione non è specificato, viene considerata l'ultima connessione aperta. Se nessuna connessione è aperta, la funzione prova a stabilire una connessione come se mysql_connect() fosse richiamata senza argomenti ed usa questa. Il paramentro opzionale modo_risultato può essere MYSQL_USE_RESULT e MYSQL_STORE_RESULT. Il valore predefinito MYSQL_STORE_RESULT, così il risultato è bufferato. Vedere anche mysql_unbuffered_query() per la controparte di questo comportamento.
Mysql_query() ritorna FALSE se la query non è andata a buon fine o se non si hanno i permessi per accedere alle tabelle, TRUE in caso di successo, tranne che per SELECT, SHOW, EXPLAIN o DESCRIBE per le quali restituisce l'id_connessione.

Se la query ha successo, si può richiamare mysql_num_rows() per scoprire quante righe sono state restituite da un'istruzione SELECT o mysql_affected_rows() per scoprire quante righe sono state coinvolte da un'istruzione DELETE, INSERT, REPLACE o UPDATE.

Solo per le istruzioni SELECT, SHOW, DESCRIBE o EXPLAIN, mysql_query() restituisce un *nuovo identificativo di risultato* che si può passare a mysql_fetch_array() e ad altre funzioni che si occupano dei risultati delle tabelle. Quando si conclude il trattamento del risultato, si possono liberare le risorse associate ad esso richiamando mysql_free_result(). Comunque la memoria sarà liberata automaticamente al termine dell'esecuzione dello script.

resource mysql_unbuffered_query (string query[,resource id_connessione [, int modo_risultato]])
mysql_unbuffered_query() invia query a MySQL senza caricare nel buffer le righe risultanti come fa mysql_query(). Da una parte, questo risparmia un considerevole quantità di memoria con le query che producono risultati di grandi dimensioni. Dall'altra parte, si può iniziare l'elaborazione dei risultati immediatamente dopo che la prima riga è stata recuperata: non si deve attendere finché la quety sia completamente eseguita. Quando si usano diverse connessioni a database, si deve specificare il parametro opzionale id_connessione.
Il parametro opzionale modo_risultato puòessere
MYSQL_USE_RESULT
MYSQL_STORE_RESULT (valore predefinito) il risultato non è bufferato

bool mysql_close ([resource identificativo_connessione])
Chiude la connessione con il server MySQL associata all'identificativo di connessione specificato. Restituisce TRUE in caso di successo, altrimenti FALSE. Se non è specificato identificativo_connessione, viene usata l'ultima connessione aperta. L'uso di **mysql_close()** non è normalmente necessario, dal momento che le connessioni non persistenti sono chiuse automaticamente alla fine dell'esecuzione dello script. mysql_close() non chiude le connessioni persistenti create da mysql_pconnect().

int mysql_errno ([resource identificativo_connessione])
MySQL non visualizza gli errori è cura del programmatore tramite questa funzione di intercettare il numero di errore dall'ultima funzione MySQL, oppure 0 (zero) se nessun errore è intercorso. Notare che questa funzione restituisce solo il codice errore *della più recente funzione MySQL eseguita*, quindi prima di usarla bisogna controllarne il valore prima di chiamare un'altra funzione.

string mysql_error ([resource identificativo_connessione])

Usata per ottenere la descrizione dell'errore. Se non vi sono errori si avrà una stringa vuota.

```
<?php
   mysql_connect (  "localhost", "utente", "password" );
   mysql_select_db ( "scuolo" );
   echo mysql_errno( ) . ": " . mysql_error(). "\n";
   mysql_select_db  ( "scuola" );
   mysql_query  (  "SELECT * FROM studenti"  );
   echo mysql_errno() . ": " . mysql_error() . "\n";
?>
```

Per le funzioni di errore se non è specificato l'argomento identificativo_connessione viene usata l'ultima connessione aperta.

int mysql_num_rows (resource risultato)

Restituisce il numero di righe di una tabella risultante da una SELECT. Per recuperare il numero di righe coinvolte dalle query INSERT, UPDATE o DELETE si può usare mysql_affected_rows().

```
<?php
        $connessione = mysql_connect ( "localhost", "utente_mysql", "password_mysql" );
        mysql_select_db ( "database",  $connessione );
        $risultato = mysql_query ( "SELECT * FROM tabella", $connessione );
        $num_righe = mysql_num_rows ( $risultato );
        echo "$num_righe Righe\n";
?>
```

int mysql_affected_rows ([resource identificativo_connessione])

restituisce il numero di righe coinvolte nell'ultima query INSERT, UPDATE o DELETE associata a identificativo_connessione. Se non è specificato l'identificativo di connessione, viene considerata l'ultima connessione aperta con mysql_connect().

mysql_affected_rows() funziona solo con le istruzioni che modificano righe, quindi non con la SELECT. Se l'ultima query è una DELETE senza clausola WHERE, saranno cancellate tutte le righe dalla tabella ma la funzione restituirà zero.

Usando UPDATE, MySQL non aggiornerà le colonne nelle quali il nuovo valore è uguale al vecchio valore, quindi la funzione non restituira le righe interessate alla query ma solo quelle realmente modificate. Se l'ultima query fallisce, questa funzione restituisce -1.

L'

```
<?php
 //    Un esempio di query di eliminazione
  mysql_pconnect("localhost", "utente", "password") or die ("Connessione non riuscita: " .
         mysql_error() );
  // questo dovrebbe restituire il numero di record eliminati
  mysql_query ( "DELETE FROM mia_tabella WHERE id  <  10" );
  printf ( "Records eliminati: %d  ", mysql_affected_rows() );
  // senza la clausola WHERE nell'istruzione DELETE,    dovrebbe restituire 0
  mysql_query ( "DELETE FROM mia_tabella" );
  printf ("Record eliminati: %d\n", mysql_affected_rows());
?>
```

output sarà: Records eliminati: 10 Records eliminati: 0

```
<?php
  // Query di aggiornamento
   mysql_pconnect( "localhost", "utente", "password" ) or die  ( "Connessione non riuscita: " .
         mysql_error() );
   // aggiornamento dei record
   mysql_query( "UPDATE mia_tabella SET used = 1 WHERE id  <  10" );
   printf ( "Record aggiornati: %d\n", mysql_affected_rows() );
   mysql_query( "COMMIT" );
?>
```

L' output sarà: Record aggiornati: 10

bool mysql_free_result (resource id_risultato)
Libera tutta la memoria associata all'identificativo del risultato. Può esser richiamata in caso il risultato di una query sia una tabella di grandi dimensioni. Restituisce TRUE in caso di successo, altrimenti FALSE .

L'output dei risultati
Quando lanciamo una query tramite il php su un databse mysql abbiamo tre diversi modi di utilizzare il risultato ottenuto mediante le funzioni mysql_fetch_row, mysql_fetch_assoc e mysql_fetch_array.
Supponendo in ognuno dei casi di avere eseguito la query e avere il risultato nella variabile $res. Inoltre supponiamo che la nostra query sia la seguente:

SELECT attributo1, attributo2, attributo3 FROM miaTabella

Con la funzione **mysql_fetch_row** i risultati verranno traseferiti riga per riga in un array ordinato:

```
while (  $row  =  mysql_fetch_row ( $res ) ) {
    echo  $row[ 0 ];        // stampa attributo1
    echo  $row[ 1 ];        // stampa attributo2
    echo  $row[ 2 ];        // stampa attributo3
}
```

Gli attributi sono restituiti nell'ordine in cui vengono selezionati. Il che vuol dire che se la query fosse stata:
SELECT attributo2, attributo2, attributo3 FROM miaTabella
echo $row[0];   Restituirebbe attributo2.

Con la **mysql_fetch_assoc** invece restituisce un array associativo i cui indici sono i nomi dei vettori:

In in questo caso non dobbiamo

```
while (  $row = mysql_fetch_assoc ( $res ) ) {
    echo $row [ 'attributo1' ];    // stampa
attributo1
    echo $row [ 'attributo2' ];    // stampa
attributo2
    echo $row [ 'attributo3' ];    // stampa
```

preoccuparci dell'ordine in cui mettiamo i campi nella query in quanto ci riferiamo a ciascun attributo utilizzando il nome dello stesso (o l'alias).
La funzione **mysql_fetch_array** torna una array che contiene sia dei riferimenti associativi ai campi sia dei riferimenti numerici, in pratica si comporta come come mysql_fetch_row e mysql_fetch_assoc insieme. In questo caso potremo scrivere:

La funzione mysql_fetch_array accetta

```
while (  $row = mysql_fetch_array ( $res ) ) {
    echo $row [ ‘attributo1′];      // stampa
attributo1
    echo $row [ 2 ];                 // stampa
attributo3
    echo $row [ 1 ];                // stampa
```

un secondo parametro opzionale, il result type tale parametro può assumere 3 valori: MYSQL_ASSOC, MYSQL_NUM e MYSQL_BOTH.
Se il parametro e MYSQL_ASSOC il risultato sarà un array associativo come in mysql_fetch_assoc, se il paramrtro è MYSQL_NUM è come se utilizzassimo mysql_fetch_assoc, se non passimo il parametro o passiamo MYSQL_BOTH il risultato è quello descritto sopra.

array mysql_fetch_row (resource risultato)
Ad ogni chiamata la funzione restituisce una riga del risultato di una query in un array con *indice numerico*. Ogni colonna del risultato è memorizzata in un elemento dell’ array, partendo dall'elemento di indice 0. se non ci sono più righe restituisce FALSE

array mysql_fetch_assoc (resource risultato)
Simile alla precedente ma l’ array è associativo. Se due o più colonne del risultato hanno gli stessi nomi di attributo, l'ultima colonna avrà la precedenza. Per accedere alle altre colonne con lo stesso nome, si deve accedere al risultato con l'indice numerico usando mysql_fetch_row() oppure aggiungere degli alias.

```
<?php
  $conn = mysql_connect(  "localhost", "utente", "password"  );
  if  (!$conn) {
     echo "Impossibile connettersi al DB: " . mysql_error();
     exit;
  }
  if  ( !mysql_select_db ("mio_nome_db") ) {
     echo "Impossibile selezionare mio_nome_db: " . mysql_error();
     exit;
  }
  $sql = "SELECT  id as  id_utente,  nome, stato  FROM  tabella   WHERE stato_utente = 1";
  $risultato = mysql_query (  $sql  );
  if (  !$risultato  ) {
     echo  "Esecuzione della  query ( $sql ) fallita : " . mysql_error();
     exit;
  }
  if (  mysql_num_rows(  $risultato  ) == 0 ) {
     echo "Nessuna riga trovata";
     exit;
  }
  while ( $riga  =  mysql_fetch_assoc ( $risultato ) ) {
     echo $riga ["id_utente"];
     echo $riga ["nome"];
     echo $riga ["stato"];
  }
 mysql_free_result($risultato);
?>
```

array mysql_fetch_array (resource risultato [, int tipo_risultato])

Oltre a memorizzare i dati del risultato in array con indice numerico, questa li memorizza anche con indici associativi usando i nomi dei campi come chiavi. Se due o più colonne del risultato hanno gli stessi nomi di attributo, l'ultima colonna avrà la precedenza. Per accedere alle altre colonne con lo stesso nome, si deve usare l'indice numerico della colonna o farne un alias. Per le colonne-alias, non si può accedere al contenuto con il nome della colonna originale (in questo esempio si usa 'attributo').
Il secondo argomento opzionale tipo_risultato in mysql_fetch_array() è una costante e può assumere i seguenti valori: MYSQL_ASSOC, MYSQL_NUM e MYSQL_BOTH.

```
<?php
   mysql_connect ( "localhost", "utente", "password" ) or
              die  ( "Connessione non riuscita: " . mysql_error() );
   mysql_select_db( "mio_db" );
   $result = mysql_query ( "SELECT id, nome FROM mia_tabella" );
   while ( $riga = mysql_fetch_array  (  $risultato,  MYSQL_NUM) ) {
       printf ("ID: %s Nome: mysql_free_result(  $risultato  ); %s", $riga[0],  $riga[1] );
    }
?>
```

object mysql_fetch_object (resource risultato)
E' simile a mysql_fetch_array(), con una differenza*: viene restituito un oggetto invece che un array*. Indirettamente, questo significa che si ha solo accesso ai dati attraverso i nomi dei campi e non attraverso il loro indice (i numeri sono nomi di proprietà illeciti).

```
<?php
  echo $riga -> campo;         //   valido
  echo $riga -> 0;             //   non  valido
?>
<?php
       mysql_connect("hostname", "utente", "password");
       mysql_select_db ( $db );
       $risultato = mysql_query( "select * from tabella" );
       while (  $riga  =  mysql_fetch_object(  $risultato  ) ) {
          echo $riga -> id_utente;
          echo $riga -> nome_intero;
       }
       mysql_free_result( $risultato );
?>
```

object mysql_fetch_field (resource risultato [, int indice_attributo])
Restituisce un oggetto contenente le informazioni di un attributo. Se non è specificato l'indice dell' attributo, viene considerato il successivo attributo non ancora recuperato da mysql_fetch_field(). Le proprietà dell'oggetto sono:

nome	descrizione
name	nome della attributo
table	nome della tabella a cui appartiene la attributo
max_length	massima lunghezza della attributo
not_null1	se la attributo non può essere NULL
primary_key1	se la attributo è una chiave primaria
unique_key1	se la attributo è una chiave unica
multiple_key1	se la attributo è una chiave non-unica
numeric1	se la attributo è numerica
blob1	se la attributo è un BLOB
Type	tipo della attributo
unsigned1	se la attributo è senza segno
zerofill1	se la attributo è completata da zeri

array mysql_fetch_lengths (resource risultato)
Memorizza in un array le lunghezze di ogni colonna dell'ultima riga restituita da mysql_fetch_row(), mysql_fetch_array() e mysql_fetch_object() in un array, iniziando con un indice pari a 0 oppure FALSE in caso di errore.

bool mysql_data_seek (resource identificativo_risultato, int numero_riga)
Sposta il cursore all'interno del risultato di un offset ben preciso (numero_riga) . La successiva chiamata a mysql_fetch_row() dovrebbe restituire questa riga. numero_riga deve essere un valore nell'intervallo che va da 0 a **mysql_num_rows** - 1. La funzione mysql_data_seek() può essere usata solo con mysql_query(), non con mysql_unbuffered_query(). Restituisce TRUE in caso di successo, FALSE altrimenti.

```
<?php
  $connessione = mysql_pconnect ("localhost", "utente", "password" )  or die  ("Connessione
       non riuscita: " . mysql_error() );
   mysql_select_db   (  "samp_db" )  or die ("Selezione del database non riuscita: " .
        mysql_error() );
    $query = "SELECT cognome, nome FROM amici";
    $risultato = mysql_query  (  $query  )  or die  ("Query fallita: " . mysql_error());
    //               caricamento righe in ordine inverso
    for (  $i = mysql_num_rows( $risultato ) - 1;   $i   >=   0;  $i--  )
      {
             if   ( !mysql_data_seek (  $risultato, $i )  )
              {
                echo "Non si può eseguire il seek alla riga $i: " .  mysql_error() . "\n";
                continue;
              }
             if(  !($riga = mysql_fetch_object (  $risultato  ) )  )       continue;
             echo "$riga -> cognome $riga -> nome<br />\n";
      }
    mysql_free_result($risultato);
    mysql_close( $connessione );
?>
```

resource mysql_list_dbs ([resource identificativo_connessione])
restituisce il puntatore ad una lista contenete i database disponibili in Mysql. Usare la funzione mysql_tablename() o mysql_fetch_array() per esplorare il risultato.

```
<?php
 $connessione = mysql_connect ( 'localhost', 'utente_mysql',  'password_mysql' );
 $lista_db = mysql_list_dbs ( $connessione );
  while (  $riga = mysql_fetch_object(  $lista_db  ) ) {
          echo $riga->Database . "\n";
       }
?>
```

string mysql_db_name (resource risultato, int i [, mixed attributo])
Ritorna il nome del database presente alla i-esima riga, di una lista risultato della chiamata a mysql_list_dbs().

```
<?php
 error_reporting (  E_ALL  );
  mysql_connect (  'dbhost',  'nome',  'password'  );
  $lista  =  mysql_list_dbs ( );
  $i = 0;
  $conteggio = mysql_num_rows ( $lista );
   while ( $i  <  $conteggio ) {
       echo mysql_db_name ( $lista_db, $i ) . "\n";
       $i++;
    }
?>
```

string mysql_field_name (resource risultato, int indice_ attributo)
restituisce il nome dell' attributo specificato dall'indice. risultato deve essere un identificativo di risultato valido e indice_attributo è lo spiazzamento numerico del attributo. indice_attributo inizia da 0.

```
<?php
       /* La tabella utenti è costituita da tre attributi: id_utente,  nome_utente ,  password */
       $connessione = mysql_connect('localhost', "utente_mysql",  "password_mysql");
       mysql_select_db($nome_db, $connessione)  or die ("Errore nella selezione di $dbname: " .
       mysql_error());
       $risultato = mysql_query("select * from utenti", $connessione);
       echo mysql_field_name (  $risultato, 0 ) . "\t";
       echo mysql_field_name (  $risultato, 2 );
?>
```

L' output sarà: id_utente password

string mysql_field_flags (resource risultato, int indice_attributo)
restituisce i flag dell'attributo specificato. I flag sono separati da uno spazio in modo da poterli isolare usando *explode().*
Vengono restituiti i seguenti flag: "*not_null*"*,*"*primary_key*"*, "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment", "timestamp".*

int mysql_field_len (resource risultato, int indice_attributo)
restituisce la lunghezza dell'attributo specificato.

int mysql_field_seek (resource risultato, int indice*_attributo *)
Esegue il seek ad uno specifico indice di attributo. Se la successiva chiamata a mysql_fetch_field() non include un indice di attributo, quello specificato in mysql_field_seek() viene restituito.

string mysql_field_table (resource risultato, int indice_attributo)
Ottiene il nome della tabella in cui si trova l' attributo specificato.

string mysql_field_type (resource risultato, int indice_attributo)
E' simile alla funzione mysql_field_name(). Gli argomenti sono identici, ma viene restituito il tipo del' attributo.

```
<?php
        mysql_connect ( "localhost", "utente_mysql", "password_mysql" );
        mysql_select_db ( "mysql" );
        $risultato = mysql_query ( "SELECT * FROM func" );
        $campi = mysql_num_fields( $risultato );
        $righe= mysql_num_rows( $risultato );
        $tabella = mysql_field_table( $risultato, 0 );
        echo "La tabella'".$table."' ha ".$fields." attributi e ".$righe."    righe\n";
        echo "La tabella ha i seguenti attributi:\n";
        for ($i = 0; $i < $campi; $i++) {
           $tipo = mysql_field_type($risultato, $i);
           $nome = mysql_field_name($risultato, $i);
           $lung = mysql_field_len($risultato, $i);
           $flag = mysql_field_flags($risultato, $i);
           echo $tipo." ".$nome." ".$lung." ".$flag."\n";
        }
        mysql_free_result ( $risultato );
        mysql_close();
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:
La tabella 'func' ha 4 attributi e 1 riga
La tabella ha i seguenti attributi:
string name 64 not_null primary_key binary
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum

string mysql_get_client_info (void)
mysql_get_client_info() restituisce una stringa che rappresenta la versione della libreria client.

```
<?php
      printf ("Informazioni sul client MySQL: %s\n",
      mysql_get_client_info());
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:
Informazioni sul client MySQL: 3.23.39

string mysql_get_host_info ([resource identificativo_connessione])
restituisce una stringa che descrive il tipo di connessione in uso per identificativo_connessione, includendo il nome dell'host del server. Se identificativo_connessione è omesso, sarà usata l'ultima connessione aperta.

```
<?php
  mysql_connect ( "localhost", "utente_mysql", "password_mysql" ) or
            die  (  "Connessione non riuscita: " . mysql_error()  ) ;
  printf ( "Informazioni sull'host MySQL: %s\n", mysql_get_host_info()  );
?>
```

L' output sarà: Informazioni sull'host MySQL: Localhost via UNIX socket

string mysql_get_server_info ([resource identificativo_connessione])
restituisce la versione del server usato dalla connessione identificativo_connessione. Se identificativo_connessione è omesso, sarà usata l'ultima connessione aperta.

```
<?php
        mysql_connect("localhost", "utente_mysql", "password_mysql") or
           die("Connessione non riuscita: " . mysql_error());
        printf ("Versione server MySQL: %s\n", mysql_get_server_info());
?>
```

L' output sarà: Versione server MySQL: 4.0.1-alpha

string mysql_info ([resource identificativo_connessione])
restituisce informazioni dettagliate relative all'ultima query relativa identificativo_connessione specificato. Se non è specificato identificativo_connessione, viene considerata l'ultima connessione aperta. mysql_info() restituisce una stringa per tutte le istruzioni elencate di seguito. Per tutte le altre restituisce FALSE. Il formato della stringa dipende dall'istruzione data.
INSERT INTO ... SELECT ...
String format: Records: 23 Duplicates: 0 Warnings: 0
INSERT INTO ... VALUES (...),(...),(...)...
String format: Records: 37 Duplicates: 0 Warnings: 0
LOAD DATA INFILE ...
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
ALTER TABLE
String format: Records: 60 Duplicates: 0 Warnings: 0
UPDATE
String format: Rows matched: 65 Changed: 65 Warnings: 0
I numeri sono indicati solo a titolo esemplificativo; i loro valori corrispondono alla query.

int mysql_insert_id ([resource identificativo_connessione])
restituisce il valore generato da una query su un attributo con AUTO_INCREMENT, 0 se la query non ha generato un valore. Se è necessario salvare il valore per usarlo in seguito, assicurarsi di richiamare *mysql_insert_id()* immediatamente dopo la query che ha generato il valore.

```
<?php
   mysql_connect( "localhost", "utente_mysql", "password_mysql" ) or
              die ( "Connessione non riuscita: " . mysql_error() );
   mysql_select_db("mio_db");
   mysql_query( "INSERT INTO mia_tabella (prodotto) VALUES ('kossu')" );
   printf ("L'ultimo record inserito ha l'identificativo %d\n", mysql_insert_id());
?>
```

resource mysql_list_fields (string database, string tabella [,resource id_connessione])
Recupera le informazioni relative ad una data tabella. Gli argomenti sono il nome del database ed il nome della tabella. Viene restituito un risultato puntatore che può essere usato con mysql_field_flags(), mysql_field_len(), mysql_field_name() e mysql_field_type().

```
<?php
       $connessione = mysql_connect ( 'localhost',  'utente_mysql',   'password_mysql' );
       $campi = mysql_list_fields ( "itis",  "studenti",  $connessione );
       $colonne = mysql_num_fields ( $campi );
       for ( $i = 0;  $i < $colonne;  $i++ ) {
          echo mysql_field_name ( $campi, $i ) . "\n";
       }
?>
```

L' output sarà :
attributo1

attributo2
attributo3

resource mysql_list_tables (string database [,resource identificativo_connessione])
Accetta in ingresso il nome di un database e restituisce il puntatore ad una lista contenente i nomi delle tabelle presenti. Per esplorare la lista si può usare mysql_tablename o qualsiasi funzione per i risultati delle tabelle, come mysql_fetch_array().

```
<?php
   $nome_db = 'nome_db_mysql';
   if  ( !mysql_connect ( 'host_mysql', 'utente_mysql', 'password_mysql' ) ) {
         print 'Connessione a mysql non riuscita';
         exit;
        }
   $risultato = mysql_list_tables  (  $nome_db );
   if (  !$risultato ) {
        print "Errore database, Impossibile elencare le tabelle\n";
        print 'Errore MySQL: ' . mysql_error();
        exit;
    }
   while (  $riga = mysql_fetch_row( $risultato ) ) {
      print "Tabella: $riga[0]\n";
   }
   mysql_free_result($risultato);
?>
```

int mysql_num_fields (resource risultato)
mysql_num_fields() restituisce il numero degli attributi di un risultato.
Vedere anche: mysql_select_db(),mysql_query(), mysql_num_rows() e mysql_fetch_field().

resource mysql_pconnect ([string server [, string nome_utente [, string password [, int flag_client]]]])
E' simile a mysql_connect() con due principali differenze:

- quando si connette, la funzione tenta innanzitutto di trovare una connessione (persistente) già aperta avente gli stessi host, username e password. Se viene trovata ne viene restituito un identificativo anziché aprirne una nuova.
- quando l'esecuzione dello script termina la connessione al server SQL non viene chiusa. La connessione rimane invece aperta per usi futuri (mysql_close() non chiude le connessioni stabilite da mysql_pconnect()).

Questo tipo di link è perciò chiamato 'persistente'.

bool mysql_ping ([resource identificativo_connessione])
Restituisce TRUE se la connessione al server è funzionante, altrimenti FALSE. Se la connessione è caduta, viene tentata una riconnessione automatica. Può essere usata dagli script che rimangono inattivi per un lungo periodo per controllare se il server ha chiuso la connessione o meno e riconnettersi se necessario.

mixed mysql_result (resource risultato, int attributo [, mixed attributo])
Restituisce il valore di una cella da un risultato MySQL. L'argomento attributo può essere:

- l'indice dell' attributo
- il nome dell' attributo
- nome della tabella ed il nome del attributo separati da un punto

(nome_tabella.nome_attributo)

- Se il nome della colonna ha un alias ('select foo as bar from...'), si può usare l'alias.

Quando si lavora con un risultato di grandi dimensioni, è opportuno usare delle funzioni che caricano l'intera riga. Poiché queste funzioni restituiscono i contenuti di celle multiple in una chiamata a funzione, sono MOLTO più veloci di mysql_result(). Notare anche che specificare l' indice di un attributo è molto più veloce che specificare un argomento del tipo nome_di_attributo o nome_tabella.attributo.
Le chiamate a mysql_result() non dovrebbero essere mescolate con chiamate ad altre funzioni che hanno a che fare con i risultati.
Alternative raccomandate per alte prestazioni: mysql_fetch_row(), mysql_fetch_array() e mysql_fetch_object().

string mysql_stat ([resource identificativo_connessione])
Restituisce l'attuale stato del server.
Nota: mysql_stat() attualmente restituisce solo le seguenti informazioni: uptime, thread, query, tabelle aperte, tabelle svuotate e query al secondo. Per una lista completa delle altre variabili di stato si usi il comando SQL SHOW STATUS.

```
<?php
        $connessione = mysql_connect ( 'localhost', "utente_mysql", "password_mysql" );
        $stato = explode ( ' ', mysql_stat ( $connessione ) );
        print_r ( $stato );
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:

```
Array
(
   [0] => Uptime: 5380
   [1] => Threads: 2
   [2] => Questions: 1321299
   [3] => Slow queries: 0
   [4] => Opens: 26
   [5] => Flush tables: 1
   [6] => Open tables: 17
   [7] => Queries per second avg: 245.595
)
```

string mysql_tablename (resource risultato, int i)
restituisce il nome della i-esima tabella presente nella lista ottenuta come risultato della chiamata alla funzione mysql_list_tables().
La funzione mysql_num_rows() può essere usata per determinare il numero di tabelle nel risultato.

```
<?php
   mysql_connect ( "localhost", "utente_mysql", "password_mysql" );
    $risultato = mysql_list_tables ( "mio_db" );
    for ( $i = 0; $i < mysql_num_rows($risultato); $i++ )
          printf ("Tabella: %s\n", mysql_tablename ( $risultato, $i ));
    mysql_free_result($risultato);
?>
```

int mysql_thread_id ([resource identificativo_connessione])
mysql_thread_id() restituisce l'attuale thread ID. Se La connessione è persa a ci si riconnette con mysql_ping(), il thread ID cambia. Questo significa che non si dovrebbe ottenere il thread ID e memorizzarlo per impieghi successivi. Si dovrebbe rilevare il thread ID quando è necessario.

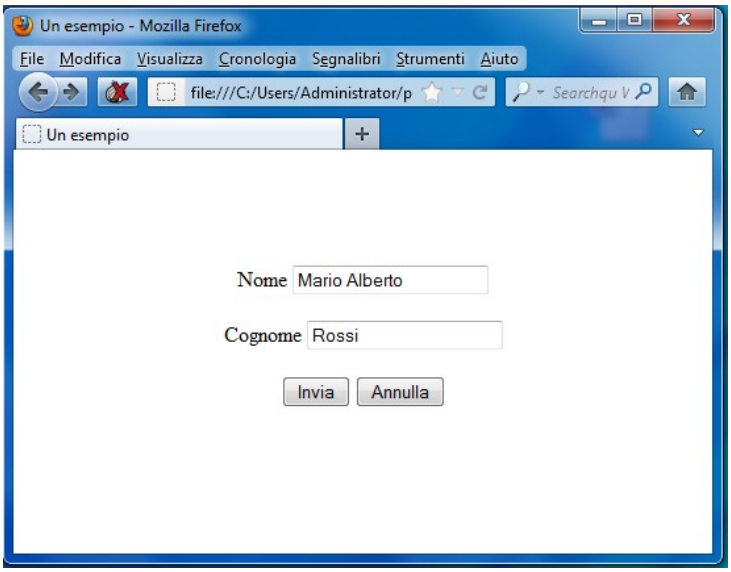
```
<?php
  $connessione  =  mysql_connect ( 'localhost', 'utente_mysql',  'password_mysql' );
  $thread_id  =  mysql_thread_id ( $connessione );
  if ( $thread_id ){
        printf (  "L'attuale thread è %d\n",  $thread_id  );
  }
?>
```

L' output sarà ad esempio : L'attuale thread è 73

Interazione con l'utente tramite l'HTML

L'interazione con l'utente può avvenire con l'ausilio dei FORM dell'HTML, che come abbiamo visto in precedenza permettono la costruzione di un' interfaccia grafica, formata da campi di input e da pulsanti. Tramite questa interfaccia, l'utente può inserire valori ed inviarli ad uno script PHP residente su di un server Web. Il programma PHP elabora i dati ed invia i risultati al client costruendo con essi una pagina HTML (le pagine html vengono dette dinamiche perché cambiano al cambiare dei risultati). Il client attraverso il browser visualizza la pagina ricevuta.
Questo è un esempio di un form HTML che passa due campi, *nome* e *cognome* ad uno script php:

```
<HTML>
<HEAD><TITLE>Un esempio</TITLE></HEAD>
<BODY>
<BR><BR><BR><BR><CENTER>
  <FORM ACTION="http://192.168.2.233/rispondi.php" METHOD = GET >
      Nome      <INPUT TYPE = text NAME = nome      SIZE = ”20"><BR><BR>
      Cognome <INPUT TYPE = text NAME = cognome SIZE = ”30"><BR><BR>
      <INPUT TYPE = submit VALUE = Invia>
      <INPUT TYPE = reset VALUE = Annulla>
  </FORM>
</CENTER>
</BODY>
</HTML>
```



Quando si fa click sul pulsante Invia, il browser richiama lo script php rispondi.php , indicato come valore dell'attributo ACTION. Il browser aggiunge automaticamente alla richiesta tutti i valori dei campi presenti nel form html. Per ogni campo crea una coppia *attributo-valore* formata da

< nome campo - valore campo >

- nome campo è il valore dell'attributo NAME ;
- valore campo è la stringa inserita dall'utente nella corrispondente casella.

L'attributo **METHOD** del form ha il compito di indicare al browser quale metodo deve utilizzare per inviare i campi del form al server web. Le modalità sono **GET** e **POST.**

Nella modalità **GET** i parametri vengono aggiunti automaticamente dal browser come parte dell'indirizzo e *vengono visualizzati insieme* all'URL. Nell'esempio precedente supponendo che l'utente abbia inserito nei due campi di input Mario Alberto Rossi l' indirizzo diventerà:

http://192.168.2.233/rispondi.php**?**nome=Mario**+**Alberto**&**cognome=Rossi

L'indirizzo vero e proprio viene separato dalle coppie attributo-valore dal **?,** mentre le coppie sono separate fra loro da **&**. Lo spazio bianco viene sostituito dal **+.**

Il metodo GET pone due problemi:

- *Sicurezza*: la stringa è visibile, quindi se trasmettiamo una password essa è visibile a tutti..
- *Dimensioni* : il metodo ha un limite di lunghezza della stringa inviata per cui in caso ad esempio di una TEXT AREA è sconsigliato.

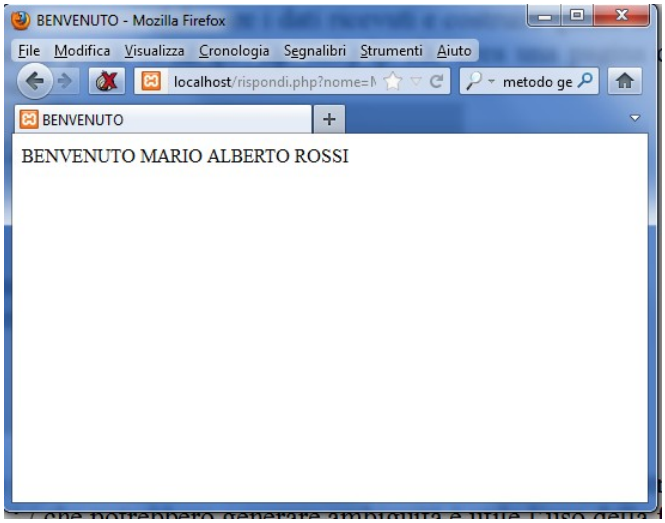
Nella modalità **POST** i parametri non vengono trasmessi come parte dell'URL e sono invisibili all'utente. Con questo metodo si superano i problemi posti dal metodo GET, che generalmente viene utilizzato quando vi è la necessità di consentire il salvataggio dell'intero indirizzo nel bookmarks da parte dell'utente.

I parametri vengono passati allo script php a seconda del metodo usato, tramite gli array associativi **$_GET** o **$_POST**. I valori sono gli elementi dell'array corrispondenti all'indice costiutuito dal nome campo. Nel nostro esempio i valori inseriti tramite il form HTML sono accessibili nello script PHP tramite:

```
$_GET ["nome"]          // contiene Mario Alberto
$_GET ["cognome"]       // contiene Rossi
```

Quindi il PHP può costruire elaborare i dati ricevuti e costruire quindi la pagina HTML di risposta. Di seguito viene proposto lo script *rispondi.php* che crea una pagina di benvenuto in HTML ed inviarla al browser.

```
<HTML>
<HEAD><TITLE>BENVENUTO</TITLE></HEAD
>
<BODY>
 <?php
        $nome      = $_GET ["nome"] ;
        $cognome = $_GET ["cognome"];
        echo "Benvenuto " . $nome . $cognome;
 ?>
</BODY>
```



NOTA: Quando si passa una URL quale valore all'interno di una querystring o quando si passano caratteri come “ ‘ / che potrebbero generare ambiguità è utile l’uso della funzione **urlencode** che effettua l'encoding della stringa passata come argomento. Vediamo un esempio:

```
$url = http://www.pippo.it/php/";
echo urlencode ( $url );
```

L'output prodotto sarà: **http%3A%2F%2Fwww.pippo.it%2Fphp%2F**
Vediamo un esempio:

```
$nextpag = "http://www.pippo.it";
echo "<a href=\"vai.php?url=" . urlencode($nextpag) . "\">Clicca qui!</a>";
```

dove il file vai.php redireziona l'utente alla URL specificata nella querystring.

Nel tuo istituto è presente un server su cui potrai esercitarti sia nella programmazione client-side che in quella server-side. Su questo server sono installati i seguenti servizi:

- server http Apache;
- estensioni PHP e PERL;
- Java server Tomcat;
- Server FTP Filezilla.
- Server di posta elettronica Mercury

Per poter interagire con il server dovrai usare l'indirizzo IP **192.168.2.231**.
I tuoi testi HTML e i tuoi programmi PHP una volta relizzati in locale sul tuo PC dovranno esser trasferiti via FTP sul server per essere provati. Le possibili modalità sono due:

1. il client **ftp.exe** testuale;
2. il client ftp **filezilla client** grafico.

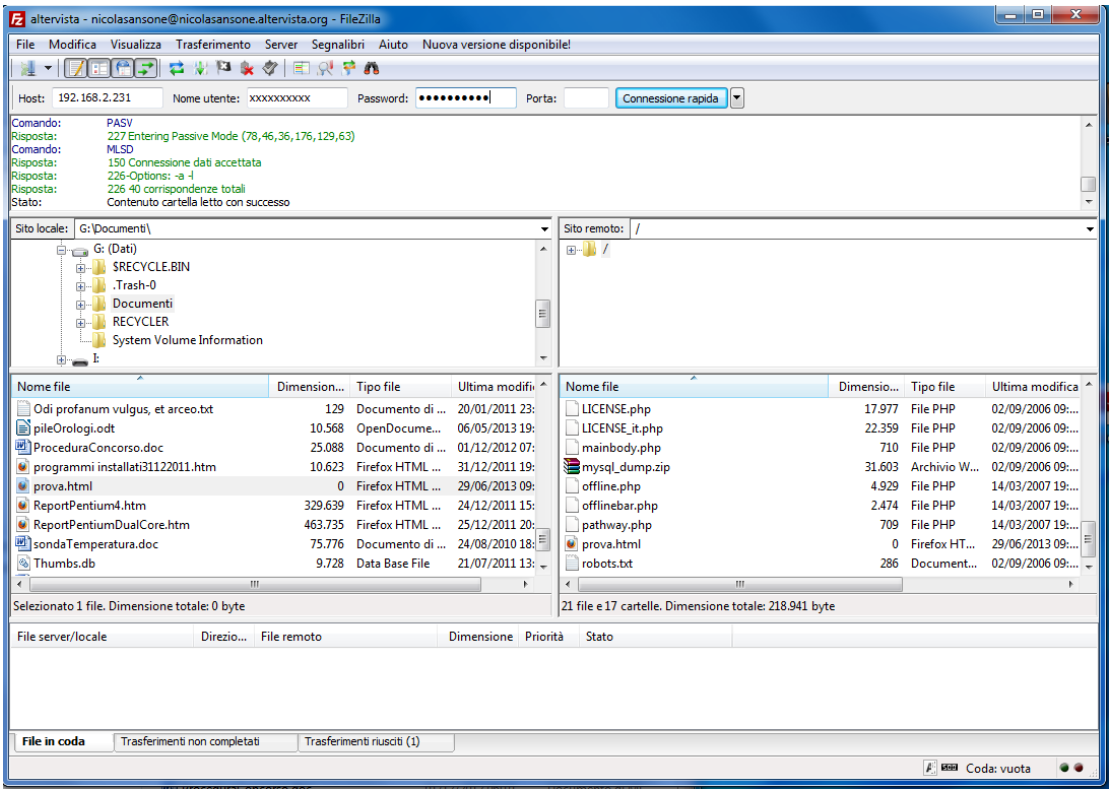
Quello che segue è un esempio di utilizzo di ftp.exe

Dalla finestra comandi digitare **ftp 192.168.2.231** e poi **INVIO**

Connesso a 192.168.2.231
220-FileZilla Server version 0.9.23
220-
220-Benvenuto nel server FTP dell' ITIS Pacinotti
220 Buon lavoro
Utente: xxxxxxx *inserisci l' identificativo assegnato dal docente*
Password: xxxxxxx *inserisci la password assegnata dal docente*
230 Logged on
ftp>

per fare l'upload ***put*** *file-locale file-remoto*
per fare il download ***get*** *file-remoto file-locale*

per l'help di tutti i comandi consentiti digitare al prompt ftp>***help***
L'esempio che segue è il trasferimento dei file HTML e PHP tramite filezilla client



Una volta trasferiti i file è sufficiente sulla barra degli indirizzi del tuo browser digitare

http://192.168.2.231/classe/tuonome/xxxx.html