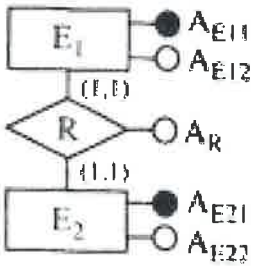
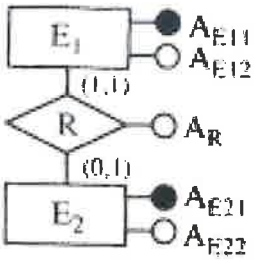
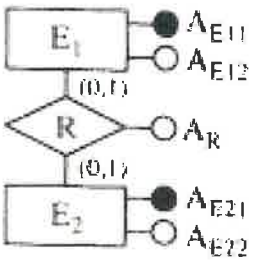
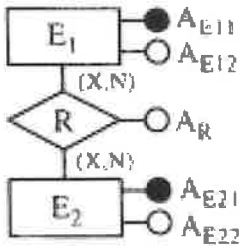
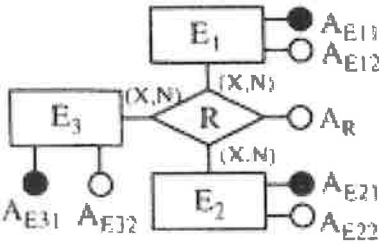
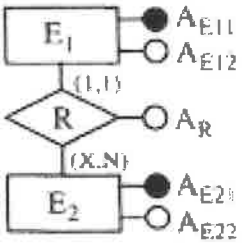
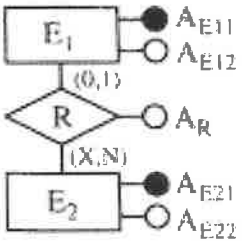
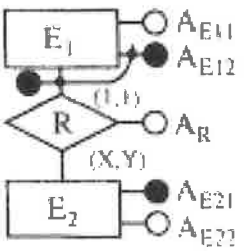
MANUALE DI INFORMATICA

Sommario

Progettazione database

Regole di derivazione

Tipologia	Concetto iniziale	Risultati possibili
Associazione uno a uno con partecipazione obbligatoria per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ Oppure: $E_2(\underline{A_{E21}}, A_{E22}, \underline{A_{E11}}, A_R)$ $E_1(\underline{A_{E11}}, A_{E12})$
Associazione uno a uno con partecipazione opzionale per una entità		$E_1(\underline{A_{E11}}, A_{E12}, \underline{A_{E21}}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$
Associazione uno a uno con partecipazione opzionale per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22}, A^*_{E11}, A^*_R)$ Oppure: $E_1(\underline{A_{E11}}, A_{E12}, A^*_{E21}, A^*_R)$ $E_2(\underline{A_{E21}}, A_{E22})$ Oppure: $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$

Tipologia	Concetto iniziale	Risultati possibili
Associazione binaria molti a molti		$E_1(\underline{A_{E11}},A_{E12})$ $E_2(\underline{A_{E21}},A_{E22})$ $R(\underline{A_{E11},A_{E21}},A_R)$
Associazione ternaria molti a molti		$E_1(\underline{A_{E11}},A_{E12})$ $E_2(\underline{A_{E21}},A_{E22})$ $E_3(\underline{A_{E31}},A_{E32})$ $R(\underline{A_{E11},A_{E21},A_{E31}},A_R)$
Associazione uno a molti con partecipazione obbligatoria		$E_1(\underline{A_{E11}},A_{E12},A_{E21},A_R)$ $E_2(\underline{A_{E21}},A_{E22})$
Associazione uno a molti con partecipazione opzionale		$E_1(\underline{A_{E11}},A_{E12})$ $E_2(\underline{A_{E21}},A_{E22})$ $R(\underline{A_{E11}},A_{E21},A_R)$ Oppure: $E_1(\underline{A_{E11}},A_{E12},A^*_{E21},A^*_R)$ $E_2(\underline{A_{E21}},A_{E22})$
Associazione con identificatore esterno		$E_1(\underline{A_{E12},A_{E21}},A_{E11},A_R)$ $E_2(\underline{A_{E21}},A_{E22})$

Algebra Relazionale

l'algebra relazionale rappresenta un insieme di operazioni formali che consentono di manipolare e interrogare i dati memorizzati nelle tabelle (o relazioni).

Le operazioni dell'algebra relazionale si distinguono in operatori unari, che agiscono su una singola relazione, e operatori binari, che richiedono due relazioni come input.

Operatori unari

Gli operatori unari lavorano su una sola relazione alla volta e consentono di selezionare, ristrutturare o rinominare i dati in base a determinate condizioni.

Nome operazione	Simbolo	Operatore	Descrizione
Selezione (σ)	$R = \sigma_P A$	WHERE	La selezione estrae un sottoinsieme "orizzontale" della relazione.
Proiezione (π)	$R = \pi_L A$	SELECT	La proiezione estrae un sottoinsieme "verticale" della relazione.

Operatori binari

Gli operatori binari operano su due relazioni alla volta e sono fondamentali per confrontare, combinare o differenziare insiemi di dati provenienti da tabelle diverse.

Nome operazione	Simbolo	Operatore	Descrizione
Prodotto cartesiano ($\times$)	$R = A \times B$	CROSS JOIN	Il prodotto cartesiano di due relazioni A e B genera tutte le coppie formate da una tupla di A e una tupla di B.
Join ($\bowtie$)	$R = A \bowtie B$	JOIN	Il join di due relazioni A e B genera tutte le coppie formate da una tupla di A e una tupla di B "semanticamente legate".
Unione ($\cup$)	$R = A \cup B$	UNION	L'unione di due relazioni A e B seleziona tutte le tuple presenti in almeno una delle due relazioni. Le relazioni A e B devono avere lo stesso schema
Intersezione ($\cap$)	$R = A \cap B$	INTERSECT	L'intersezione di due relazioni A e B seleziona tutte le tuple presenti in entrambe le relazioni. le relazioni A e B devono avere lo stesso schema (numero e tipo degli attributi)
Differenza ($-$)	$R = A - B$	EXCEPT	tutte le tuple appartenenti ad A che non appartengono a B
Divisione ($/$)	$R = A/B$	* NOT EXISTS EXCEPT GROUP BY HAVING	La divisione della relazione A per la relazione B genera una relazione R • avente come schema schema(A) - schema(B) • contenente tutte le tuple di A tali che per ogni tupla (Y:y) presente in B esiste una tupla (X:x, Y:y) in A

*non esiste una vera e propria corrispondenza

SQL

DDL (Data Definition Language)

Creazione di un database

```
CREATE SCHEMA <nomeDatabase>
DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;
```

- SCHEMA e DATABASE sono **sinonimi;**
- Per **distruggere** un database si utilizza DROP SCHEMA <nomeDatabase>;
- Per **modificare** un database si utilizza ALTER SCHEMA <nomeDatabase> DEFAULT COLLATE ...

Creazione di tabelle

```
CREATE TABLE <nomeDatabase>.<nomeTabella> (
    <Id> INT UNSIGNED NOT NULL AUTO_INCREMENT,
    <attributo1> <TIPO_DATO> (<dimensione>) <NULL>|<NOT NULL>,
    <attributo2> <TIPO_DATO> (<dimensione>) <NULL>|<NOT NULL>,
    <attributo3> <TIPO_DATO> (<dimensione>) <NULL>|<NOT NULL>,
    <attributo4> <TIPO_DATO> (<dimensione>) <NULL>|<NOT NULL>,
    PRIMARY KEY (<id>)
    FOREIGN KEY (<attributo2>)  REFERENCES <altra_tabella> (<chiave_altra_tabella>)
    ...opzionale..
    ON UPDATE CASCADE
    ON DELETE CASCADE
    )
```

Tipo di dati numerici		
BOOLEAN	Valore logico	True<>0, False=0
INTEGER	Numero intero	Ne esitono di vari tipi.
DECIMAL(M,D)	Numero con la virgola	A packed "exact" fixed-point number. M is the total number of digits (the precision) and D is the number of digits after the decimal point (the scale). Usarlo per i campi valuta, ad esempio DECIMAL(13,4)
FLOAT(M,D)	Numero con la virgola	A small (single-precision) floating-point number
DOUBLE(M,D)	Numero con la virgola	A normal-size (double-precision) floating-point number.

Tipo di dati caratteri		
CHAR(M)	Alfanumerico	A fixed-length string that is always right-padded with spaces to the specified length when stored. 0<M<255
VARCHAR(M)	Alfanumerico	A variable-length string. M represents the maximum column length in characters. 0<M<65535
BLOB(M)	Binary large object	A BLOB column with a maximum length of 65535 bytes. Character set = binary.
TEXT(M)	Nonbinary strings (character strings)	A TEXT column with a maximum length of 65535 characters.
ENUM	Esempio: size ENUM('x-small', 'small', 'medium', 'large', 'x-large')	String object with a value chosen from a list of permitted values
SET		String object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created

Tipo di dati tempo		
DATE	Data	'YYYY-MM-DD'
DATETIME	Data e ora	'YYYY-MM-DD HH:MM:SS[.fraction]'
TIME	Ora	'HH:MM:SS[.fraction]'

Alter Table

Add

Aggiunge una colonna ad una tabella

```
ALTER TABLE <nomeTabella>
ADD <nomeColonna> <TIPO_DATO>;
```

Drop

Elimina una specifica colonna in una tabella

```
ALTER TABLE <nomeTabella>
DROP COLUMN <nomeColonna> <TIPO_DATO>;
```

Modify

Modifica il nome e il tipo di dato di una specifica colonna in una tabella

```
ALTER TABLE <nomeTabella>

MODIFY COLUMN <nomeColonna> <TIPO_DATO>;
```

Rename

```
ALTER TABLE <nomeTabella>

....

RENAME TABLE <nomeTabellaCorrente>  TO <nuovoNomeTabella>;

...

RENAME COLUMN <NomeCampoCorrente>  TO <NuovoNomeCAmpo>;
```

DML (Data Manipulating Language)

Inserimento di dati in una tabella

```
INSERT INTO <nomeTabella>(<nomeColonna>[, <nomeColonna>...]) VALUES (<valore>[, <valore>...]);
```

Se nella nostra tabella sono presenti campi **opzionali**, per non inserire dati in quella colonna è sufficiente non specificare il nome della colonna corrispondente. In questo caso l'attributo sarà inizializzato a **NULL**.

N.B. NULL è **diverso** da stringa vuota ("").

Aggiornamento di tabelle

```
UPDATE <nomeTabella>
SET <nomeColonna> = ...
WHERE <nomeColonna> [ = | < | > ] ... ;
```

Cancellazione di righe di una tabella

```
DELETE FROM <nomeTabella>
WHERE <nomeColonna> [ = | < | > ] ... ;
```

Una DELETE senza la clausola WHERE cancella interamente i dati della tabella.

QL (Query Language)

La clausola SELECT (interrogazione a una base di dati)

```
SELECT { * | <nomeColonna> [, <nomeColonna>] ... }
FROM <nomeTabella>
WHERE <condizione>;
```

- Con il predicato ALL si richiede che nel risultato dell'interrogazione siano incluse **tutte** le righe, anche quelle duplicate. È il default ed in genere è omesso.
- Con il predicato DISTINCT si richiede che le righe duplicate nel risultato siano ridotte ad **una**:

```
SELECT DISTINCT { * | <nomeColonna> [, <nomeColonna>] ... }
FROM <nomeTabella>;
```

- AS serve a dare un **alias** al nome delle colonne di una tabella, quando eseguiamo una SELECT. Se l'alias contiene spazi, si usano i doppi apici (""):

```
SELECT <nomeColonna> AS alias
FROM <nomeTabella>;
```

JOIN

Le JOIN si usano quando vogliamo selezionare i dati provenienti da colonne di **due o più tabelle** associate tramite delle **chiavi esterne** (**FK**). Esistono due tipi di JOIN:

- INNER JOIN
- OUTER JOIN (RIGHT JOIN o LEFT JOIN): includono anche quelle righe di una delle tabelle che non hanno corrispondenza nell'altra.

Inner Join

Supponiamo di avere due tabelle associate tramite una chiave esterna (foreign key). La INNER JOIN seleziona le righe che hanno valori corrispondenti in **entrambe** le tabelle. Le parole chiavi INNER JOIN definiscono la **congiunzione** fra le tabelle. La condizione di congiunzione (quali colonne congiungono le due tabelle?) è scritta dopo la parola chiave ON.

```
SELECT ...
FROM <nomeTabella1> INNER JOIN <nomeTabella2>
ON <nomeTabella1>.id1 = <nomeTabella2>.id2;
```

Left join

Seleziona tutte le righe della tabella che sta a **sinistra** della join, anche quelle che non hanno una corrispondenza nella tabella che sta dall'altra parte.

```
SELECT ...
FROM <nomeTabella1> LEFT JOIN <nomeTabella2>
ON <nomeTabella1>.id1 = <nomeTabella2>.id2;
```

Right join

Seleziona tutte le righe della tabella che sta a **destra** della join, anche quelle che non hanno una corrispondenza nella tabella che sta dall'altra parte.

```
SELECT ...
FROM <nomeTabella1> RIGHT JOIN <nomeTabella2>
ON <nomeTabella1>.id1 = <nomeTabella2>.id2;
```

Possiamo selezionare solo le righe della tabella che sta a sinistra o a destra che non hanno una corrispondenza nella tabella che sta dall'altra parte:

```
SELECT ...
FROM <nomeTabella1> LEFT|RIGHT JOIN <nomeTabella2>
ON <nomeTabella1>.id1 = <nomeTabella2>.id2
WHERE id2 IS NULL;
```

Ordinamento (ORDER BY)

Se l'ordinamento è crescente NULL compare all'inizio della sequenza. La clausola va posizionata **in fondo** a tutte le altre clausole della query.

```
SELECT <nomeColonna>
FROM <nomeTabella>
ORDER BY <nomeColonna> ASC; → crescente ... Possiamo omettere
```

```
SELECT <nomeColonna>
FROM <nomeTabella>
ORDER BY <nomeColonna> DESC; → decrescente
```

Funzioni di aggregazione

Le funzioni di aggregazione agiscono sui valori contenuti in insiemi di righe, agiscono sui valori di una colonna e restituiscono **un solo valore**. Possono comparire solo nelle clausole SELECT e HAVING.

- COUNT() **conta** il numero delle occorrenze selezionate dall'interrogazione.

  ```
  SELECT COUNT(<nomeColonna>)
  FROM <nomeTabella>
  WHERE ... ;
  ```

  COUNT(*) equivale a contare il numero delle righe di una tabella.

- SUM() restituisce la **somma** dei valori contenuti in una colonna. I valori NULL sono trascurati. L'argomento della funzione SUM può essere un'espressione.

  ```
  SELECT SUM(<nomeColonna>)
  FROM <nomeTabella>
  WHERE ... ;
  ```

- MIN() e MAX() restituiscono rispettivamente il valore **minimo** e **massimo** assunti da una colonna. Funzionano anche per i campi di tipo carattere. I valori NULL sono ignorati e possono avere come argomento un'espressione.

```
SELECT MIN(<nomeColonna>)
FROM <nomeTabella>
WHERE ... ;
SELECT MAX(<nomeColonna>)
FROM <nomeTabella>
WHERE ... ;
```

- AVG() restituisce la **media** dei valori contenuti in una colonna. I valori NULL sono trascurati.

```
SELECT AVG(<nomeColonna>)
FROM <nomeTabella>
WHERE ... ;
```

Raggruppamento (GROUP BY)

La clausola GROUP BY **raggruppa** un insieme di righe aventi lo **stesso valore** nelle colonne indicate dalla clausola. Produce una riga di risultati per ogni raggruppamento. Se viene inserita una funzione di aggregazione, per ciascuna riga del risultato viene prodotto un valore di raggruppamento. Nella clausola SELECT possono comparire solo i campi presenti in GROUP BY e funzioni di aggregazione. I NULL vengono raggruppati.

```
SELECT <nomeColonna>, SUM(<nomeColonna>)
FROM <nomeTabella>
GROUP BY <nomeColonna>;
```

Condizioni sui raggruppamenti (HAVING)

La clausola HAVING si può utilizzare solo in presenza di GROUP BY e permette di visualizzare solo i raggruppamenti che soddisfano le condizioni scritte di seguito ad essa. In genere è utilizzata per controllare il valore restituito dalle funzioni di aggregazione.

```
SELECT <nomeColonna>, SUM(<nomeColonna>)
FROM <nomeTabella>
GROUP BY <nomeColonna>
HAVING COUNT(<nomeColonna>) [ = | < | > ] ... ;
```

LIMIT

La seguente istruzione riduce il numero di righe:

LIMIT n (seleziona le prime n righe)

LIMIT k,n (seleziona dal record k (ricordo che il primo record è zero) e ne estrae n)

Quadro riassuntivo delle diverse clausole che possono apparire nel comando SELECT

- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY
- LIMIT

Le condizioni di ricerca

- L'operatore BETWEEN permette di verificare dei valori all'interno di un determinato **intervallo**. I valori possono essere numeri, testo o date. I valori iniziali e finali sono **inclusi**.

  SELECT <nomeColonna>
  FROM <nomeTabella>
  WHERE <nomeColonna> BETWEEN ... AND ... ;

- L'operatore IN permette di specificare **più valori** in una clausola WHERE.

  SELECT <nomeColonna>
  FROM <nomeTabella>
  WHERE <nomeColonna> IN (...);

  L'operatore NOT IN è l'inverso dell'IN.

- Non è possibile verificare se un valore è NULL con gli operatori di confronto come =, <, >... In alternativa si utilizzano gli operatori IS NULL e IS NOT NULL.

  SELECT <nomeColonna>
  FROM <nomeTabella>
  WHERE <nomeColonna> IS NULL | IS NOT NULL ;

- L'operatore LIKE viene utilizzato in una clausola WHERE per analizzare il **contenuto parziale** di una stringa. Esistono due **caratteri jolly** spesso utilizzati insieme all'operatore LIKE: il segno "%" rappresenta zero, uno o più caratteri qualsiasi; il segno "_" rappresenta un singolo carattere qualsiasi.

  SELECT <nomeColonna>
  FROM <nomeTabella>
  WHERE <nomeColonna> LIKE "C%" ; → controlla se la stringa inizia con la lettera "C"

  L'operatore NOT LIKE è l'inverso del LIKE.

- L'operatore AND mostra un record se **tutte** le condizioni separate dall'AND sono TRUE.
- L'operatore OR mostra un record se **almeno una delle** condizioni separate dall'OR è TRUE.

Interrogazioni annidate (subquery)

Il comando SELECT interno (detto **subquery**) deve comparire incluso in una coppia di parentesi tonde, senza punto e virgola. Servono per risolvere i problemi dove compaiono confronti con una funzione di aggregazione. Deve esserci coerenza tra WHERE e SELECT annidata.

SELECT <nomeColonna>
FROM <nomeTabella>
WHERE <nomeColonna> [= | < | >]
(SELECT <nomeColonna>|MAX|MIN|COUNT|SUM|AVG(<nomeColonna>) FROM <nomeTabella>);

Operatori ANY e ALL

Gli operatori ANY e ALL sono usati insieme alla clausola WHERE oppure HAVING. ANY restituisce true se **uno qualsiasi** dei valori della subquery soddisfa la condizione. ALL restituisce true se **tutti** i valori della subquery soddisfano la condizione.

```
SELECT <nomeColonna>
FROM <nomeTabella>
WHERE <nomeColonna> [ = | < | > ] ALL|ANY(
SELECT <nomeColonna> FROM <nomeTabella> WHERE ... );
```

Operatore EXISTS

L'operatore EXISTS restituisce un valore **boolean** (true o false) ed è utilizzato per verificare se una subquery restituisce o meno dei risultati.

```
SELECT <nomeColonna>
FROM <nomeTabella>
WHERE EXISTS (
SELECT <nomeColonna> FROM <nomeTabella> WHERE ... );
```

Le view

L'utente può operare sulle view con i comandi SQL visti in precedenza, proprio come se fosse una tabella. Le view sono **dinamiche**: ogni modifica effettuata sulle tabelle primarie è disponibile nella view e viceversa. Fatta eccezione delle view che contengono funzioni di aggregazione, le quali non sono aggiornabili.

- Creare una view:

  ```
  CREATE VIEW <nomeView> AS (SELECT ... );
  ```

- Utilizzare una view:

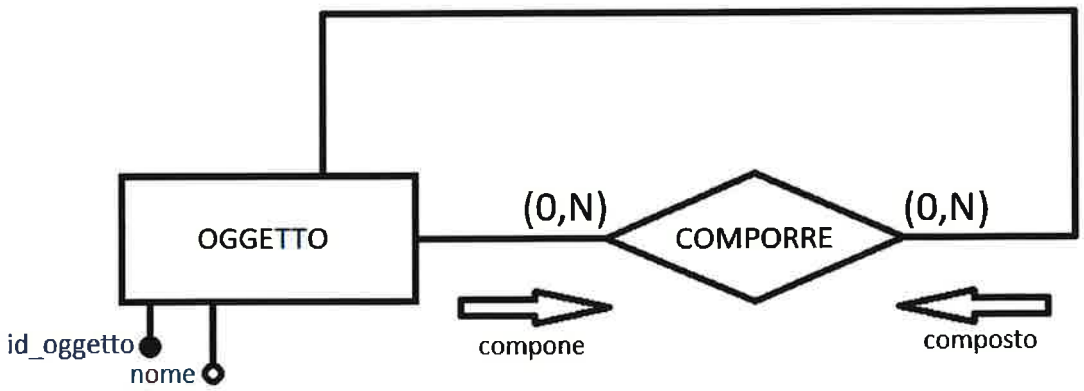
  ```
  SELECT <nomeColonna> FROM <nomeView> WHERE ... ;
  ```

- Eliminare una view:

  ```
  DROP VIEW <nomeView>;
  ```

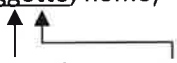
SELF JOIN

Una SELF JOIN è una join regolare, ma la tabella è **unita a sé stessa**.



Ogni oggetto può essere composto da uno o più oggetti e ogni oggetto può comporre uno o più oggetti.

oggetto(id_oggetto, nome)



aggregato(composto, componente)

Per trovare tutti gli oggetti composti da altri oggetti scriveremo:

```
SELECT composto.nome, componente.nome
FROM aggregato AS A INNER JOIN oggetto AS composto
ON A.composto=composto.id_oggetto
INNER JOIN oggetto AS componente
ON A.componente=componente.id_oggetto
```

Gestione delle date in MySQL

I tipi di dato "data" e "tempo" di MySQL sono i seguenti:

TIPO	Formato standard	Note
DATE	YYYY-MM-DD	
TIME	hh:mm:ss	
DATETIME	YYYY-MM-DD hh:mm:ss[.fraction]	Aggiornamento automatico
TIMESTAMP	YYYY-MM-DD hh:mm:ss[.fraction]	Aggiornamento automatico
YEAR	YYYY	

La funzione DATE_FORMAT

La funzione DATE_FORMAT(<data>, <formato>) formatta il valore specificato in "date" secondo la stringa specificata nel formato. La stringa è composta dagli "specificatori" di formato. I più utilizzati sono:

- %a Nome abbreviato del giorno (Sun ... Sat)
- %b Nome abbreviato del mese (Jan ... Dec)
- %c Mese in formato numerico (1 ... 12)
- %m Mese in formato numerico con prefisso "0" (01 ... 12)
- %d Giorno del mese in formato numerico con prefisso "0" (00 ... 31)
- %e Giorno del mese in formato numerico (0 ... 31)
- %H Ora (24h) (00 ... 23)
- %h Ora (12h) (01 ... 12)
- %i Minuti in formato numerico (00 ... 59)

- %M Nome del mese (January ... December)
- %s Secondi (00 ... 59)
- %T Tempo (24h) (hh:mm:ss)
- %W Nome del giorno della settimana (Sunday ... Saturday)
- %Y Anno in formato numerico, quattro cifre
- %y Anno in formato numerico, due cifre

La funzione STR_TO_DATE

La funzione STR_TO_DATE(<stringa>, <formato>) è l'inverso del DATE_FORMAT. Restituisce DATETIME se il formato contiene sia una parte data che una parte tempo, oppure restituisce DATE o TIME se il formato contiene solo una parte data o solo una parte tempo. Se la stringa non corrisponde ad alcun formato, la funzione restituisce NULL e produce un warning.

TIMESTAMP vs DATETIME

MySQL converte i TIMESTAMP dal "current time zone" all'UTC (Universal Time Code) prima di memorizzare il dato su disco ed effettua la conversione inversa quando si recupera il dato (ciò non avviene per i DATETIME). Di default il "current time zone" per ciascuna connessione è quello del server. Ma il time zone può anche essere impostato "per-connection".

Per creare delle colonne che hanno come default il valore della data e ora attuale e si aggiornano automaticamente alla data e all'ora attuale quando una colonna qualsiasi viene aggiornata:

```
CREATE TABLE <nomeTabella> (
    <attributo1> TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;
    <attributo2> DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP;
);
```

La funzione CURDATE() restituisce la data corrente "aaaa-mm-dd"

La funzione CURTIME() restituisce l'ora corrente "HH:MM:SS"

La funzione NOW() restituisce data e orario AAAA-MM-DD HH:MM:SS

YEAR(data) estrae l'anno

MONTH(data) estrae il mese

DAY(data) estrae giorno del mese

WEEKDAY(data) estrae giorno settimana 0 (Lunedì)....6 (Domenica)

ADDDATE(data, numero giorni) aggiunge un numero di giorni ad una data ...e quindi restituisce la data finale

SUBDATE(data, numero giorni) sottrae un numero di giorni ad una data ...e quindi restituisce la data finale

La funzione DATEDIFF

La funzione DATEDIFF(<data1>, <data2>) restituisce la differenza espressa in giorni tra due date.

DCL (Data Control Language)

Il Data Control Language (**DCL**) è un linguaggio usato in SQL per fornire o revocare agli utenti i **permessi** necessari per poter utilizzare i comandi di DDL e DML, oltre agli stessi comandi di DCL.

Comandi standard del DCL

- CREATE ROLE <ruolo> [, <ruolo>];
- DROP ROLE <ruolo> [, <ruolo>];
- CREATE USER '<nomeUtente>'@'<indirizzoDB>'
  IDENTIFIED BY '<password>'
  [DEFAULT ROLE <ruolo>];
- ALTER USER '<nomeUtente>'@'<indirizzoDB>'
  IDENTIFIED BY '<password>';
- DROP USER '<nomeUtente>' [, '<nomeUtente>'];

GRANT

Il comando GRANT fornisce uno o più permessi a un determinato utente su un determinato oggetto del database.

```
GRANT <privilegi>
ON <oggetto>
TO <utente>
[WITH GRANT OPTION]
```

Privilegi: può essere la parola ALL (per garantire tutti i permessi all'utente) o uno o più permessi specifici del database, come: CREATE DATABASE, SELECT, INSERT, UPDATE, DELETE e CREATE VIEW.

Oggetto: può essere qualsiasi oggetto del database, ovvero: il database stesso, una tabella, una view...

Utente: può essere un qualsiasi utente del database.

Se si specifica **WITH GRANT OPTION**, l'utente avrà la possibilità di assegnare gli stessi privilegi anche ad altri utenti, o eventualmente revocarli.

REVOKE

Il comando REVOKE revoca uno o più permessi a un determinato utente su un determinato tipo di oggetti.

```
REVOKE [GRANT OPTION FOR] <permessi>
ON <oggetto>
FROM <utente>
[CASCADE | RESTRICT]
```

Permessi: specifica i permessi da togliere all'utente.

Oggetto: può essere qualsiasi oggetto del database, ovvero: il database stesso, una tabella, una view...

Utente: può essere un qualsiasi utente del database.

GRANT OPTION FOR specifica che non si intende eliminare il privilegio in sé, ma il diritto di un certo utente di accordare o revocare tale permesso ad altri.

CASCADE indica che bisogna eliminare gli oggetti ai quali nessun utente ha più il permesso di accedere. **RESTRICT**, che è il valore predefinito, indica che tali oggetti devono essere preservati.

DENY

Il comando DENY impedisce esplicitamente a un utente di ricevere una particolare autorizzazione.

```
DENY <privilegi>
ON <oggetto>
TO <utente>
```

I parametri per il comando DENY sono identici a quelli utilizzati per il comando GRANT.

Programmazione di applicazioni web (PHP)

La sintassi del linguaggio PHP

Tag di apertura e chiusura

```
<?php
  ...
?>
```

Ogni istruzione in PHP termina con il simbolo **punto e virgola** (;).

Strutture di controllo, commenti e operatori

Le strutture di controllo del flusso di esecuzione sono praticamente le stesse dei linguaggi C/C++ e Java:

- if-else;
- switch-case;
- while;
- do-while;
- for;
- foreach.

I **commenti** di un programma PHP seguono le regole dei linguaggi C/C++ e Java con l'uso di /*...*/ per commenti su più righe e // per commentare un'unica riga.

I principali **operatori algebrici** e di **assegnamento** del PHP sono simili a quelli dei linguaggi C/C++ e Java con l'aggiunta di quelli che permettono la concatenazione di stringhe.

Operatore	Significato
=	Assegnamento
+	Somma
-	Sottrazione
*	Moltiplicazione
**	Potenza
/	Divisione
%	Resto
++	Autoincremento
--	Autodecremento
+=	Assegnamento combinato a somma
-=	Assegnamento combinato a differenza
*=	Assegnamento combinato a moltiplicazione
/=	Assegnamento combinato a divisione
%=	Assegnamento combinato a resto
.	Concatenazione di stringhe (punto)
.=	Assegnamento combinato a concatenazione di stringhe

Operatore	Significato
==	Uguaglianza
!=	Disuguaglianza
===	Identità (uguaglianza e stesso tipo)
<	Minore
>	Maggiore
<=	Minore o uguale
>=	Maggiore o uguale
<=>	*Spaceship*

Per la formulazione di **espressioni logiche**, il linguaggio PHP prevede i classici operatori:

Operatore	Significato
&& and	AND
|| or	OR
^ xor	XOR
!	NOT

Output in PHP

L'output in PHP può essere gestito tramite due istruzioni identiche:

echo "Hello world!"; è analogo a print "Hello world!";

La differenza sostanziale è che echo è in grado di stampare uno o più valori, mentre print ne gestisce uno alla volta:

echo "Uno", "Due", "Tre"; è analogo a print "Uno Due Tre";

All'interno di uno script può presentarsi la necessità di interrompere il flusso delle istruzioni a seguito del verificarsi di determinate condizioni: in questi casi si ricorre alle istruzioni **die** o **exit** (identiche sia nella sintassi che nelle funzionalità). Determinano quindi l'arresto immediato del flusso delle istruzioni e consentono di inserire nella pagina web un eventuale messaggio.

```
<?php
    ...
    if(condizione) {
        die("Errore nell'applicazione!");
    };
    ...
?>
```

Le variabili nel linguaggio PHP

Le variabili hanno il nome che inizia sempre con il simbolo "$" e non necessitano di una dichiarazione esplicita del tipo. Esso viene deciso in fase di esecuzione da PHP a seconda del contesto in cui viene utilizzata quella variabile.

Tipi scalari

Tipo di dato	Descrizione
`boolean`	Rappresenta un valore booleano che può assumere i valori `true` o `false`.
`integer`	La sua rappresentazione dipende dalla piattaforma del sistema in uso, ma generalmente è rappresentato su 32 byte e può contenere valori da −2147483648 a +2147483647. Oltre alla notazione in base 10 è possibile utilizzare le notazioni: • in base 8 facendo precedere il numero da uno zero: `a$=0123` (83 in base 10); • in base 16 facendo precedere il numero da 0x: `b$=0XEA2` (3746 in base 10).
`double`	È in virgola mobile su 64 bit a «precisione doppia»: i limiti teorici sono da −1.7976931348623157E+308 a −2.2250738585072014E-308 e da 2.2250738585072014E-308 a 1.7976931348623157E+308, oltre allo zero. La sintassi usata per esprimere questo tipo di numeri prevede due varianti: • notazione decimale: `a$=23.561;` • notazione esponenziale: `b$=23561e-3`.
`string`	Rappresenta una sequenza di caratteri ASCII di lunghezza massima pari a 32 cratteri. La sintassi con cui si possono esprimere le costanti di tipo stringa prevede che le medesime siano racchiuse tra apici singoli (« ' ») o doppi (« " »).

Esiste inoltre il tipo speciale **NULL**, il quale definisce una variabile senza alcun valore: la variabile esiste ma non è avvalorata.

Caratteri di escape

PHP permette di inserire nelle stringhe di caratteri sequenze di *escape*.

Sequenza di *escape*	Significato
`\n`	*Line feed*
`\r`	*Carriage Return*
`\t`	Tabulazione orizzontale
`\\`	Simbolo «\»
`\$`	Simbolo «$»
`\"`	Simbolo «"»
`\123`	Notazione per la rappresentazione ottale del codice ASCII di un carattere.
`\x12`	Notazione per la rappresentazione esadecimale del codice ASCII di un carattere.

Operatori di cast

È possibile impostare esplicitamente il tipo di una variabile o verificarne il tipo utilizzando gli **operatori di cast** riportati in tabella:

Operatore	*Cast*	Verifica
(bool), (boolean)	A booleano	`is_bool()`
(int), (integer)	A intero	`is_long()`
(real), (double)	A numero in virgola mobile	`is_float()`
(string)	A stringa	`is_string()`
(array)	Ad *array*	`is_array()`
(unset)	A `NULL`	`is_null()`

Variabili dinamiche

A volte è conveniente poter avere nomi di variabili che siano variabili. Ovvero, un nome di variabile che può essere impostato e utilizzato dinamicamente. Una variabile dinamica è impostata anteponendo due simboli di dollaro.

$alfa = "colore";

$$alfa = "Rosso";

Così facendo, la variabile $alfa conterrà il valore "colore", mentre la variabile $colore avrà come valore "Rosso".

Costanti

Per dichiarare una costante si utilizza la seguente sintassi:

define("<*NOME_COSTANTE*>", <*valore*>);

Per definire il π si può scrivere:

define("PI_GRECO", 3.14);

Gli array

In PHP gli array possono essere indicizzati sia con numeri interi positivi a partire dallo zero, sia per mezzo di una chiave di tipo stringa. In quest'ultimo caso il vettore memorizza delle associazioni chiave/valore e viene detto **array associativo**. PHP accetta che gli elementi di un vettore non siano tutti dello stesso tipo, a differenza di altri linguaggi.

Dichiarazione

La dichiarazione di un vettore in PHP avviene tramite l'utilizzo della parola chiave **array**, che accetta come argomenti una sequenza di valori oppure delle coppie *chiave => valore* nel caso degli array associativi.

- $<*nome_array*> = array(value1, value2, value3, ...);
- $<*nome_array*> = array(key1 => value1, key2 => value2, key3 => value3, ...);

  key può essere sia un numero naturale che una stringa.
- $<*nome_array*> = [value1, value2, value3, ...];

- $<*nome_array*> = value1;

  Crea l'array definendo contestualmente l'elemento di indice zero. Se l'array esiste già, viene aggiunto un nuovo elemento.

Gli elementi di un array possono essere a loro volta degli array.

Scansione dell'array

Lo scorrimento di un array in PHP può avvenire attraverso varie metodologie.

- Metodo1:

```
foreach ($<nome_array> as $valore) {
    ...
}
```

- Metodo2:

```
foreach ($<nome_array> as $chiave=>$valore) {
    ...
}
```

- Metodo3:

```
for (i=0; i<count($<nome_array>); i++) {
    ...
}
```

Array correlati al web

Per quanto riguarda le attività sul web, PHP prevede degli specifici array associativi denominati **superglobali** e dedicati a specifiche funzionalità. Questi array hanno nomi predefiniti: **$_GET**, **$_POST**, **$_COOKIE** e **$_SESSION**. I primi due sono relativi al **passaggio di dati** tra pagine web, mentre gli altri due sono dedicati alla **gestione di dati** che debbono essere comuni alle pagine esplorate durante una sessione di utilizzo del browser da parte dell'utente.

- **$_GET**: usa il metodo **GET** del protocollo **HTTP**. I dati trasferiti sono visibili nell'URL. Qualsiasi informazione inviata con la stringa d'interrogazione in una coppia *nome/valore* viene caricata nell'**array associativo** $_GET della pagina richiesta.
- **$_POST**: usa il metodo **POST** del protocollo **HTTP**. I dati trasferiti non sono visibili nell'URL. $_POST viene usato principalmente dai moduli presenti nelle pagine web, ma può essere usato indipendentemente dai **form**.

Funzioni in PHP

Funzioni per le variabili

Nome funzione	Descrizione
`empty`	Verifica se la variabile è vuota oppure no. Per «vuota» si intende che può contenere una stringa vuota o un valore numerico pari a 0, ma può anche essere non definita o essere impostata al valore `NULL` (l'eventuale indicazione di una variabile non definita, in questo caso, non genera errore). Restituisce un valore booleano. ▶
`isset`	Verifica se la variabile è definita. Una variabile risulta non definita quando non è stata inizializzata o è stata impostata al valore `NULL`. Restituisce un valore booleano.
`is_null`	Verifica se la variabile equivale a `NULL`, ma genera un errore se viene invocata su una variabile non definita. Restituisce un valore booleano.

`is_int`
`is_integer`
`is_long`
	Le tre funzioni sono equivalenti, verificano se la variabile è di tipo numerico intero. Restituiscono un valore booleano.

`is_float`
`is_double`
`is_real`
	Le tre funzioni sono equivalenti, verificano se la variabile è di tipo numerico in virgola mobile. Restituiscono un valore booleano.

`is_string`	Verifica se la variabile è una stringa. Restituisce un valore booleano.
`is_array`	Verifica se la variabile è un *array*. Restituisce un valore booleano.
`is_numeric`	Verifica se l'argomento contiene un valore numerico. È importante la distinzione fra questa funzione e *is_int()* o *is_float()*, perché queste ultime, nel caso di una stringa che contiene valori numerici, restituiscono *falso*, mentre *is_numeric()* restituisce *vero*. Restituisce un valore booleano.
`gettype`	Restituisce una stringa che rappresenta il tipo di dato dell'argomento, per esempio: **`boolean`**, **`integer`**, **`double`**, **`string`**, **`array`**.
`print_r`	Inserisce nella pagina web informazioni relative al contenuto della variabile. È utile in fase di *debug*. Se l'argomento è un *array* sono visualizzate le chiavi e i valori relativi.
`unset`	Distrugge la variabile specificata. In realtà non si tratta di una funzione, ma di un'istruzione del linguaggio. Dopo *unset()*, l'esecuzione di *empty()* o *is_null()* sulla stessa variabile restituirà *vero*, mentre *isset()* restituirà *falso*.

Funzioni per gli array

Nome funzione	Descrizione
`count`	Restituisce un intero che rappresenta il numero di elementi dell'*array*.
`sort` `rsort`	Ordinano gli elementi di un *array*, la prima in modo crescente, la seconda decrescente. Queste funzioni modificano direttamente l'*array* passato come argomento che perderà la sua composizione originale. Le chiavi vanno perse: l'*array* risultato ha chiavi numeriche a partire da 0 secondo il nuovo ordinamento.
`in_array(valore, array)`	Cerca il valore all'interno dell'*array*. Restituisce un valore booleano. ▶

`array_key_exists(valore, array)`	Cerca il valore fra le chiavi (non fra i valori) dell'*array*. Restituisce un valore booleano.
`array_search(valore, array)`	Cerca il valore nell'*array* e ne restituisce la chiave o, se la ricerca non va a buon fine, il valore `false`.
`array_merge(array, array)`	Fonde gli elementi di due (o più) *array*. Gli elementi con indici numerici vengono accodati l'uno all'altro e le chiavi rinumerate. Le chiavi associative invece vengono mantenute e, nel caso vi siano più elementi con la stessa chiave associativa, l'ultimo sostituisce i precedenti. Restituisce l'*array* risultante dalla fusione.
`array_pop(array)`	Estrae l'ultimo elemento dell'*array* da cui viene eliminato. Restituisce l'elemento estratto.
`array_push(array, valore)`	Accoda il valore indicato all'*array*. Equivale all'uso dell'istruzione di accodamento `$array[]=$valore`. Restituisce il numero degli elementi dell'*array* dopo l'accodamento.

Funzioni per data e ora

Nome funzione	Descrizione
`time()`	Fornisce il *timestamp* relativo al momento in cui viene eseguita. Restituisce un valore intero.
`checkdate(mese, giorno, anno)`	Verifica se i valori forniti costituiscono una data valida. Restituisce un valore booleano.
`mktime(ore, minuti, secondi, mese, giorno, anno)`	Sulla base dei parametri forniti restituisce un intero che rappresenta un *timestamp*. Può essere utilizzata per effettuare calcoli sulle date.

`date(formato [,timestamp])`	Considera il *timestamp* fornito (se non è indicato usa quello attuale) e fornisce una data formattata secondo le specifiche indicate nel primo parametro di tipo stringa. Per esempio le due istruzioni
	`$data = mktime(0,0,0,2,29,2012);`
	`echo date('d-m-Y',$data);`
	visualizzano 29-2-2012.

Formato data e ora

Codice	Descrizione
Y	Anno su 4 cifre
y	Anno su 2 cifre
n	Mese numerico (1-12)
m	Mese numerico su 2 cifre (01-12)
F	Mese testuale ('January' – 'December')
M	Mese testuale su 3 lettere ('Jan' – 'Dec')
d	Giorno del mese su due cifre (01-31)
j	Giorno del mese (1-31)
w	Giorno della settimana, numerico (0 = domenica, 6 = sabato)
l	Giorno della settimana, testuale ('Sunday' – 'Saturday')
D	Giorno della settimana su 3 lettere ('Sun' – 'Sat')
H	Ora su due cifre (00-23)
G	Ora (0-23)
i	Minuti su due cifre (00-59)
s	Secondi su due cifre (00-59)

Funzioni per le stringhe

Nome funzione	Descrizione
`strlen(stringa)`	Restituisce un numero intero che rappresenta la lunghezza della stringa, cioè il numero di caratteri che la compongono.

`trim(stringa)`
`ltrim(stringa)`
`rtrim(stringa)`
	Eliminano gli spazi in una stringa, rispettivamente all'inizio e alla fine, solo all'inizio, solo alla fine. Restituiscono la stringa modificata.

`substr(stringa, intero [, intero])`	Restituisce una porzione della stringa, in base al secondo parametro (che indica l'inizio della porzione da estrarre) e all'eventuale terzo parametro (che indica quanti caratteri devono essere estratti). Se il terzo parametro non viene indicato, viene restituita tutta la parte finale della stringa a partire dal carattere indicato. Per esempio, con `substr('AlfaBeta', 3, 2)` si ottiene *aB*.
`str_replace(stringa, stringa, stringa)`	Effettua una sostituzione della prima stringa con la seconda all'interno della terza. Restituisce la terza stringa modificata. Per esempio, con `str_replace('p','t','pippo')` si ottiene *titto*.
`strpos(stringa, stringa)`	Ricerca la posizione della seconda stringa all'interno della prima. Restituisce un intero che rappresenta la posizione della stringa cercata. Se la seconda stringa non è presente nella prima, restituisce il valore booleano `false`. Per esempio: `strpos('Lorenzo', 're')` restituisce 2.
`strstr(stringa, stringa)`	Ricerca la seconda stringa all'interno della prima e restituisce la parte della prima stringa a partire dal punto in cui ha trovato la seconda. Se la ricerca non va a buon fine, restituisce il valore booleano `false`. Per esempio: `strstr('Lorenzo', 're')` restituisce *renzo*.
`strtolower(stringa)` `strtoupper(stringa)`	Convertono tutti i caratteri alfabetici nelle corrispondenti lettere minuscole/maiuscole. Restituiscono la stringa modificata.
`explode(stringa, stringa [, intero])`	Scompone la seconda stringa in un *array* usando i caratteri della prima come separatori degli elementi. Il terzo parametro può servire a indicare il numero massimo di elementi che l'*array* deve contenere (se la scomposizione della stringa ne genera un numero maggiore, la parte finale della stringa sarà interamente contenuta nell'ultimo elemento). Restituisce l'*array* delle sottostringhe risultato della scomposizione. Per esempio: `explode(' ','ciao Marco')` restituisce un *array* di due elementi in cui il primo è *ciao* e il secondo *Marco*.

Funzioni per i numeri

Nome funzione	Descrizione
`intval($argomento)`	Se l'argomento della funzione è: un numero decimale lo arrotonda a intero; un numero negativo lo trasforma in positivo; altro lo trasforma in 0 (zero).
`strval($argomento)`	Restituisce il valore di una variabile *argomento* interpretato come stringa. ▶

Nome funzione	Descrizione
`intdiv($dividendo, $divisore)`	Restituisce la parte intera del quoziente della divisione tra due variabili numeriche *dividendo* e *divisore* (introdotta solo nelle versioni più recenti di PHP).
`min($arg_1, $arg_2, ..., $arg_n)` `max($arg_1, $arg_2, ..., $arg_n)`	Accettano come argomenti una lista di variabili numeriche (o *array*) e ne restituiscono rispettivamente il valore minimo o massimo. Le eventuali variabili stringa vengono valutate 0 (zero).

Funzioni definite dall'utente

Per definire una funzione in PHP si utilizza la parola chiave function e l'eventuale risultato viene restituito per il tramite della parola chiave return.

Il passaggio di parametri avviene normalmente per valore. Il passaggio per riferimento è possibile utilizzando la notazione del linguaggio C, cioè premettendo il simbolo di & al nome del parametro.

È possibile definire dei parametri con un valore di default, assegnandolo contestualmente alla definizione della funzione:

function <nomeFunzione>($<parametro1>, $<parametro2> = <*valore di default*>);

La gestione di form HTML con il linguaggio PHP

Interazione tra pagine web

Il tag HTML che ci consente di predisporre una pagina da parte dell'utente è form, che consente di specificare l'attributo *method* per scegliere tra la modalità **GET** o **POST** di inoltro dei valori inseriti/selezionati dall'utente alla pagina di destinazione specificata dall'attributo *action*.

All'interno del tag form è possibile utilizzare i tag input, textarea e select/option. Per questi tag deve essere specificato sempre l'attributo *name*, il quale consente di definire il nome a cui viene associato il valore inserito o selezionato dall'utente: se la pagina di destinazione del form contiene codice PHP, i nomi definiti sono le chiavi per gli array associativi superglobali $_GET e $_POST (a seconda del *method* prescelto). Se è possibile una selezione multipla di valori, per l'attributo *name* viene utilizzata la coppia di simboli "[]".

Il tag input non prevede un tag di chiusura. Può essere di diversi tipi (*type*).

`button`	Visualizza un pulsante normalmente utilizzato per l'invocazione di uno *script* di codice eseguito lato client.
`checkbox`	Visualizza una casella per la selezione del valore indicato nell'attributo *value*; più elementi di questo tipo aventi lo stesso valore dell'attributo *name* specificato come *array* consentono la selezione multipla di più valori.
`radio`	Visualizza un *radio button* per la selezione del valore indicato nell'attributo *value*: se esitono più elementi di questo tipo aventi lo stesso valore dell'attributo *name*, la selezione è mutuamente esclusiva.
`reset`	Visualizza un pulsante il cui effetto è quello di ripristinare i valori iniziali di tutti i componenti del *form*.
`submit`	Visualizza un pulsante il cui effetto è quello di richiedere al server la pagina specificata nell'attributo *action* del *form* passando i valori inseriti/selezionati dall'utente.
`text`	Visualizza un campo di testo; è possibile specificare l'attributo *size* per indicarne la dimensione espressa in caratteri e l'attributo *maxlength* per specificare il numero massimo di caratteri digitabili.

Il tag select visualizza un elenco di opzioni tra le quali è possibile effettuare una scelta (il numero di scelte visibili è indicato dall'attributo *size*): le singole opzioni sono specificate tramite il tag option, il cui attributo *value* identifica il valore inviato alla pagina di destinazione.

Per inizializzare gli elementi che compongono un form è possibile utilizzare l'attributo *value* per i tag input di tipo text e l'attributo *checked* impostato a true per i tag input di tipo *checkbox* e *radio*.

Form in HTML:

```
<form method="POST" action="pagina.php">
    Dato1: <input type="text" name="dato2"><br><br>
    Dato2: <input type="text" name="dato1"><br><br>
    Dato3 (selezione singola tra due opzioni): <br>
        Scelta1: <input type="radio" value="scelta1" name="dato3"><br>
        Scelta2: <input type="radio" value="scelta2" name="dato3"><br><br>
    Dato4 (scelta multipla): <br>
        A: <input type="checkbox" value="a" name="scelte[]"><br>
        B: <input type="checkbox" value="b" name="scelte[]"><br>
        C: <input type="checkbox" value="c" name="scelte[]"><br>
        D: <input type="checkbox" value="d" name="scelte[]"><br>
        E: <input type="checkbox" value="e" name="scelte[]"><br><br>
    Dato5 (selezione singola, menù a tendina): <br>
        <select name="menu" size="4">
            <option value="alfa">Alfa</option>
            <option value="beta">Beta</option>
            <option value="gamma">Gamma</option>
            <option value="delta">Delta</option>
        </select><br><br>
    <input type="reset" value="Reset">
    <input type="submit" value="Ok">
</form>
```

Lo script della pagina di destinazione *pagina.php* (che è stata specificata nell'attributo *action* del form) è il seguente:

```
<?php
    $dato1 = $_POST["dato1"] ?? $_GET["dato1"] ?? "nessuno";
    $dato2 = $_POST["dato2"] ?? $_GET["dato2"] ?? "nessuno";
    $dato3 = $_POST["dato3"] ?? $_GET["dato3"] ?? "nessuno";
    $dati4 = $_POST["scelte[]"] ?? $_GET["scelte[]"] ?? "nessuno";
    $dato5 = $_POST["menu"] ?? $_GET["menu"] ?? "nessuno";

    if($dato1 == "nessuno" || $dato1 == "" || $dato2 == "nessuno" || $dato2 == "" ||$dato3 ==
    "nessuno" || $dato3 == "" ||$dato4 == "nessuno" || $dato4 == "" ||$dato5 == "nessuno" ||
    $dato5 == "") {
        die("Attenzione! Dati non inseriti correttamente. Torna alla pagina di inserimento cliccando
            <a href='path pagina principale'>qui</a>");
    }

    echo "Primo dato: $dato1<br>";
    echo "Secondo dato: $dato2<br>";
    echo "Terzo dato: $dato3<br>";
    echo "Quarto dato:<br>";
    foreach($dati4 as $element) {
            echo "$element<br>";
    }
    echo "Quinto dato: $dato5";
?>
```

Gestione Grafica con invio posizione

```
<head>

 <meta charset="UTF-8">

 <title>Mappa Interattiva con Elementi Cliccabili</title>

</head>

<body>

 <h2>Mappa con punti interattivi</h2>


 <svg viewBox="0 0 800 600">

  <!-- Immagine mappa di sfondo -->
```

```
  <image href="https://...percorso/Immagine_di sfondo.png" x="0" y="0" height="600" width="800" />


  <!-- Elementi cliccabili rossi (con numeri) -->

  <circle cx="250" cy="150" r="10" onclick="showValue('valore1')" fill="red" stroke="black" stroke-width="1" style="cursor:pointer;" />

  <circle cx="400" cy="300" r="10" onclick=" showValue ('valore2')" fill="red" stroke="black" stroke-width="1" style="cursor:pointer;" />

 </svg>

 <script>

  function showValue (num) {

   alert("Hai cliccato sul punto: " + num);

   window.location.href = "/pagina.php?valore=" + num;  //se devo inviare in modalità get a pagina php

  }

 </script>

</body>
```

Gestione e validazione degli input nelle pagine web

Esistono delle specifiche categorie di input *type* dove il controllo formale viene effettuato direttamente dal **client**. La prima tabella mostra alcuni tipi di input che generano **campi speciali**, la seconda invece mostra alcuni attributi utilizzabili per la loro **validazione**.

Tipo di input	Esempio	Note
Indirizzi e-mail	`e-mail: <input type="email" name="email_utente">`	Controllo esistenza @.
Indirizzi web	`Sito: <input type="url" name="web_utente">`	Verifica formato URL.
Numeri	`Voto: <input type="number" name="voto" min="0" max="10">`	Possibilità di impostare attributi *min*, *max* (e *step*).
Intervalli numerici	`Voto: <input type="range" name="voto" min="1" max="10">`	Visualizzazione grafica come una barra scorrevole.
Date	`Data di nascita: <input type="date" name="data_nascita">`	Formato automatico data. Può mostrare calendario per selezione data.
Numeri telefonici	`Telefono: <input type="tel" name="telefono">`	Verifica formato numero.
Colori	`Colore: <input type="color" name="colore preferito">`	Mostra palette colori.
Password	`Password: <input type="password" name="password">`	Caratteri digitati visualizzati come *.

Attributo	Tipo	Note
autofocus	booleano	Imposta il *focus* su uno specifico elemento del *form* al momento del suo avvio.
form	generico	Permette la specifica dell'*id* del *form* a cui il campo di input fa riferimento.
placeholder	generico	Visualizza un valore di default per un campo di input, o di un'area di testo, fino a quando il campo è vuoto o non assume il *focus*.
required	booleano	Serve a rendere obbligatoria la compilazione dell'elemento a cui è applicato.
title	generico	Associa un *tooltip* informativo a un campo di input (visualizzato quando il puntatore del mouse si trova in corrispondenza del campo stesso).
autocomplete	enumerato	Consente il completamento automatico di un campo da parte del browser.
multiple	booleano	Consente l'inserimento di più valori per uno stesso input (per esempio un insieme di indirizzi e-mail a cui inviare contemporaneamente un messaggio).

Costanti predefinite per Validazione e Sanificazione PHP

FILTER_VALIDATE

Queste costanti sono utilizzate per **validare** un dato, ovvero per verificare se esso **rispetta** un determinato **formato** o **criterio di correttezza**. La validazione **restituisce** il valore **originale** se esso è **valido**, **altrimenti** restituisce "**false**". Le principali sono:

- FILTER_VALIDATE_EMAIL
- FILTER_VALIDATE_URL
- FILTER_VALIDATE_IP
- FILTER_VALIDATE_MAC
- FILTER_VALIDATE_BOOLEAN
- FILTER_VALIDATE_FLOAT
- FILTER_VALIDATE_INT

Esempio di utilizzo:

```
<?php
        $email = "john.doe@example.com";
        if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
                echo("$email is a valid email address");
        } else {
                echo("$email is not a valid email address");
        }
?>
```

FILTER_SANITIZE

Le costanti **FILTER_SANITIZE_*** vengono utilizzate per **sanificare** un dato, cioè per **rimuovere o modificare caratteri** potenzialmente **dannosi** o non validi da un input. L'obiettivo è quello di prevenire attacchi e garantire che i dati siano in un formato "pulito". Le principali sono:

- FILTER_SANITIZE_EMAIL
- FILTER_SANITIZE_URL
- FILTER_SANITIZE_NUMBER_FLOAT
- FILTER_SANITIZE_NUMBER_INT

Esempio di utilizzo:

```
<?php
        $email = "john(.doe)@exa//mple.com";
        $email = filter_var($email, FILTER_SANITIZE_EMAIL);
        echo $email;
?>
```

Gestione dei cookie e delle sessioni in PHP

Il protocollo HTTP è privo di stato (**stateless**): una volta che una pagina web viene inviata dal server al client, per il codice PHP non è più possibile accedere ai dati relativi alla pagina stessa. Lo sviluppatore deve poter memorizzare informazioni persistenti per più pagine (per esempio il login di un utente). Il linguaggio PHP ci mette a disposizione due metodi: i **cookie** e le **sessioni**.

Cookie

I cookie sono dei file che il server web memorizza nel file system della macchina client. I cookie sono caratterizzati da:

- Nome;
- Valore;
- Data e ora di scadenza;
- Percorso sul server per il quale il cookie sarà valido;
- Dominio per il quale il cookie è valido.

La funzione **setcookie()** ci permette di definire e creare un cookie. L'array superglobale **$_COOKIE** permette l'accesso ai valori dei cookie già esistenti.

Creazione di un cookie:

setcookie(string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string $domain = "" [, bool $secure = false [, bool $httponly = false]]]]]]);

Sessioni

In genere, per gestire dati persistenti, si preferisce ricorrere al meccanismo delle **sessioni**. Una sessione consiste in una serie di **accessi** a pagine PHP, effettuati in un determinato arco di tempo durante il quale viene mantenuto uno "stato" costituito da variabili **persistenti**.

A differenza di cookie, le informazioni non sono memorizzate sul computer dell'utente, bensì sul web server. Per default le informazioni scadono quando l'utente chiude il browser.

Per utilizzare le sessioni in PHP sono previste 3 funzioni base:

- **session_start()**;

- **session_unset();**
- **session_destroy()**.

La funzione session_start() **crea** una sessione. L'array associativo superglobale **$_SESSION** contiene le variabili di sessione disponibili nel nostro sito. Una volta avviata una sessione è possibile impostare e/o leggere i valori nell'array $_SESSION.

Impostazione di una sessione:

```
<?php
    session_start(); //Avvio la sessione.
?>
<!DOCTYPE html>
<html>
    <body>
        <?php
            $_SESSION["colorePreferito"] = "verde"; //Imposto una variabile di sessione.
        ?>
    </body>
</html>
```

In questo modo, per accedere alla variabile di sessione potrò scrivere:

```
echo "Il tuo colore preferito &egrave; " . $_SESSION["colorePreferito"];
```

Per modificare una variabile di sessione è sufficiente **sovrascriverla**, per **rimuovere** tutte le variabili di sessione e **distruggere** la sessione utilizziamo session_unset() e session_destroy().

Accesso a una base di dati in linguaggio PHP

Il linguaggio PHP predispone della libreria denominata ***mysqli*** (*mysql improved*) per l'**interfacciamento** con DBMS Mysql/MariaDB. La libreria espone sia di un approccio procedurale, sia di un approccio orientato agli oggetti. L'approccio procedurale è basato sulle seguenti funzioni:

Funzione	Descrizione
`mysqli_connect`	Connessione a un server DBMS specificando nome (o indirizzo IP) del computer su cui è in esecuzione, *username*, *password* e database. Restituisce un identificativo della connessione.
`mysqli_connect_errno`	Restituisce l'eventuale codice numerico di errore della connessione, il valore 0 se la connessione è stata stabilita correttamente.
`mysqli_query`	Esecuzione di un comando SQL. Per i comandi DML restituisce `true` in caso di successo, oppure `false` in caso di errore. Per le *query* restituisce un riferimento al risultato, oppure `false` in caso di errore.
`mysqli_affected_rows`	Restituisce il numero di righe coinvolte dall'ultimo comando DML eseguito, il valore –1 in caso di errore.
`mysqli_num_rows`	Restituisce il numero di righe del risultato di una *query*.
`mysqli_fetch_row`	Recupero di una riga del risultato di una *query* in forma di *array* indicizzato. Restituisce un *array*, oppure `NULL` se non vi sono ulteriori righe nel risultato.
`mysqli_fetch_assoc`	Recupero di una riga del risultato di una *query* in forma di *array* associativo avente come chiavi i nomi delle colonne. Restituisce un *array* oppure `NULL` se non vi sono ulteriori righe nel risultato.
`mysqli_fetch_array`	Recupero di una riga del risultato di una *query* in forma combinata di *array* indicizzato e associativo. Restituisce un *array* oppure `NULL` se non vi sono ulteriori righe nel risultato.
`mysqli_free_result`	Rilascio della memoria del risultato di una *query*. Non restituisce nulla.
`mysqli_close`	Chiusura della connessione al server DBMS. Restituisce `true` in caso di successo, oppure `false` in caso di errore.

Con la libreria mysqli possiamo **collegarci al database** e svolgere diverse funzioni, che vedremo nei prossimi capitoli.

Configurazione del database

Deve esistere un file per la configurazione del database, contenente le informazioni che ci servono, che sono:

- indirizzo IP del database (se si lavora in locale esso sarà 127.0.0.1, localhost);
- username di accesso;
- password di accesso;
- nome del database;
- porta del database (molto probabilmente la porta 3306).

Quindi, un ipotetico script in PHP potrebbe essere il seguente:

```
<?php
    $dbHost = "<indirizzoIP>";
    $dbUser = "<username>";
    $dbPwd = "<password>";
    $dbName = "<nomeDatabase>";
    $dbPort = "<portaDatabase>";
?>
```

Apertura e chiusura della connessione a un database

Apertura della connessione a un database

Avremo bisogno di un file che ci dia la possibilità di avviare la connessione con il database:

```
<?php
    require_once("path del file di configurazione");
    $connection = new mysqli($dbHost, $dbUser, $dbPwd, $dbName, $dbPort);
    if($connection->connect_error) {
        die("Errore di connessione al database!");
    }
?>
```

Abbiamo creato un oggetto di tipo *mysqli* contenente tutte le informazioni necessarie per il collegamento al database.

N.B. In PHP è possibile memorizzare parti di codice comune in un file che può essere incluso in un punto qualsiasi di uno script. Le funzioni *require_once*(<pathFile>) e *include_once*(<pathFile>) sono analoghe a *include*(<pathFile>) e *require*(<pathFile>), ma oltre a includere il codice PHP contenuto nel file indicato ne impediscono la replicazione, cosa che generalmente avviene quando si includono più file che a loro volta includono lo stesso file. La parola chiave *require* ha lo stesso effetto di *include*, con la differenza che la prima interrompe l'esecuzione in caso di errore.

Chiusura della connessione a un database

Il file per terminare la connessione al database è molto banale. Esso sarà strutturato in questo modo:

```
<?php
    $connection->close();
?>
```

Visualizzazione tabelle di un database

La visualizzazione delle tabelle in PHP si basa sul linguaggio **SQL**, con il comando **SELECT**. Tramite la libreria **mysqli** e le **query**, possiamo visualizzare dati sulla nostra applicazione.

```
echo "<h1>Visualizza Tabella<h1>";
require_once("path del file per l'apertura della connessione");
$query = "SELECT * FROM <nomeTabella>";
$result = $connection->query($query);
if($result->num_rows != 0) {
    echo "<table><tr><th>Attributo1</th><th>Attributo2</th></tr>";
    while($row = $result->fetch_array()) {
        echo "<tr><td>$row['<nomeColonna>']</td><td>$row['<nomeColonna>']</td></tr>";
    }
    echo "</table>";
}
$result->free();
require_once("path del file per la chiusura della connessione");
```

Un oggetto di classe *mysqli* espone i seguenti metodi:

Metodi	Descrizione
`__construct`	È il costruttore della classe. Effettua la connessione a un server DBMS di cui viene fornito come argomento il nome (o l'indirizzo IP) del computer su cui è in esecuzione, l'*username* e la *password* dell'utente e il database su cui si intende operare.
`autocommit`	Fornendo come argomento `true` o `false` abilita o disabilita la modalità **AUTOCOMMIT** del DBMS MySQL/MariaDB. Restituisce `true` in caso di successo, oppure `false` in caso di errore.
`close`	Chiude la connessione al server DBMS: restituisce `true` in caso di successo, oppure `false` in caso di errore.
`commit`	Esegue il **COMMIT** dei comandi eseguiti in precedenza. Restituisce `true` in caso di successo, oppure `false` in caso di errore.
`prepare`	Predispone un comando SQL parametrizzato per l'esecuzione (i parametri sono indicati nel comando SQL per mezzo di simboli «?»). Restituisce un oggetto di classe *mysqli_stmt*, oppure `false` in caso di errore.
`query`	Esegue un comando SQL. Per i comandi DML restituisce `true` in caso di successo, oppure `false` in caso di errore. Per le *query* restituisce un riferimento al risultato, oppure `false` in caso di errore.
`rollback`	Esegue il **ROLLBACK** dei comandi eseguiti in precedenza. Restituisce `true` in caso di successo, oppure `false` in caso di errore.

Un oggetto di classe *mysqli_result* espone le seguenti proprietà e i seguenti metodi:

Proprietà	Descrizione
`$num_rows`	Contiene il numero di righe del risultato della *query*.
`$field_count`	Contiene il numero di colonne del risultato della *query*.

Metodi	Descrizione
`fetch_all`	Restituisce un *array* in cui ogni elemento è un *array* che rappresenta una singola riga del risultato di una *query*.
`fetch_array`	Restituisce una riga del risultato di una *query* in forma combinata di *array* indicizzato e associativo, oppure `NULL` se non vi sono ulteriori righe nel risultato.
`fetch_assoc`	Restituisce una riga del risultato di una *query* in forma di *array* associativo avente come chiavi i nomi delle colonne, oppure `NULL` se non vi sono ulteriori righe nel risultato.
`fetch_row`	Restituisce una riga del risultato di una *query* in forma di *array* indicizzato, oppure `NULL` se non vi sono ulteriori righe nel risultato.
`free`	Rilascia la memoria del risultato di una *query*.

Gestione delle query in PHP con prepared statements

Supponiamo che la nostra query sia la seguente:

```
$query = "SELECT * FROM <nomeTabella> WHERE <nomeColonna>=" . $_POST["<valore>"];
```

È consigliato gestire il risultato della query con un oggetto di tipo *mysqli_stmt*. Gli oggetti di questa classe sono istanziati dal metodo *prepare* di un oggetto di classe *mysqli* e consentono di definire comandi SQL **parametrici** che possono essere eseguiti più volte, fornendo valori distinti per parametri.

Un oggetto di classe *mysqli_stmt* espone le seguenti proprietà e i seguenti metodi:

Proprietà	Descrizione
`$affected_rows`	Contiene il numero di righe coinvolte dall'ultimo comando eseguito; il valore è –1 in caso di errore.
`$errno`	Contiene l'eventuale codice numerico di errore dell'ultimo comando eseguito; il valore è 0 se il comando è stato eseguito correttamente.
`$param_count`	Contiene il numero di parametri del comando parametrico.

Metodo	Descrizione
`bind_param`	Istanzia il comando parametrico sostituendo valori effettivi ai parametri, per ogni valore è necessario definire il tipo mediante un carattere («i» = *integer*, «d» = *double*, «s» = *string*). Restituisce `true` in caso di successo, oppure `false` in caso di errore.
`close`	Conclude l'uso del comando parametrico. Restituisce `true` in caso di successo, oppure `false` in caso di errore.
`execute`	Esegue il comando parametrico con i parametri istanziati. Restituisce `true` in caso di successo, oppure `false` in caso di errore.

Quindi, per gestire il risultato della query precedente scriveremo:

```
$query = "SELECT * FROM <nomeTabella> WHERE <nomeColonna>= ?";
$stmt = $connection->prepare($query);
$stmt->bind_param("i"|"s"|"d", $_POST["valore"]);
$stmt->execute();
$result = $stmt->get_result();
```

La stringa $query, che definisce il comando parametrico argomento del metodo *prepare*, ha un simbolo "?" nella posizione in cui deve essere inserito il valore del parametro. Il primo argomento del metodo *bind_param* definisce un parametro di tipo intero, stringa o double ("i"|"s"|"d") il cui valore è il successivo parametro del metodo stesso (in questo caso è un $_POST). L'invocazione del metodo *execute* esegue il comando SQL **parametrizzato**.

Login e logout di utenti presenti nel database

Potrebbe essere presente nel database una tabella contenente i dati relativi a degli **utenti**, come ad esempio email, username, password... Con il linguaggio PHP possiamo gestire login e logout di questi utenti con il meccanismo delle **sessioni**.

Effettuare il login con PHP

Prima di effettuare il login, verifichiamo che non siano settati username e password:

```
if(!isset($_POST["<utenteInserito>"]) || !isset($_POST["<passwordInserita>"]))
```

Se l'if restituisce true, scriviamo il form per l'inserimento delle informazioni necessarie per effettuare l'accesso, altrimenti...

Supponiamo che le informazioni per effettuare il login (email, username, password...) ci siano state inviate da un form html con metodo **POST**. Il codice per effettuare il login potrebbe essere il seguente:

```
$username = $conn -> real_escape_string ($_POST["<utenteInserito>"]); //se $conn era l'oggetto
    connessione
$password = $_POST["<passwordInserita>"];
if(strlen($username) != 0 && strlen($password) != 0) {
    require_once("path del file per l'apertura della connessione");
    $query = "SELECT * FROM <nomeTabella> WHERE <nomeColonnaUtenti>=" . $username;
    $result = $connection->query($query);
    if(result->num_rows == 0) {
        echo "Utente sconosciuto: <a href='path del file per il login'>riprova</a>";
    }
    else {
        $row = $result->fetch_array();
        if(password_verify($password, $row["<nomeColonnaPassword>"])) {
            echo "Accesso effettuato, benvenuto " . $row["<nomeColonnaUsername>"];
            session_start();
            session_unset();
            session_destroy();
            session_start();
            $_SESSION["username"] = $username;
            $_SESSION["start_time"] = time();
        }
        else {
            echo "Password errata: <a href='path del file per il login'>riprova</a>";
        }
    }
```

```
$result->free();
require_once("path del file per la chiusura della connessione");
}
else {
    echo "Username o password non validi: <a href='path del file per il login'>riprova</a>";
}
```

Effettuare il logout con PHP

Una volta fatto il login, potrebbe essere utile **uscire** dall'account (**logout**). Il codice in PHP è molto semplice: la sessione creata in precedenza con il login va **distrutta**.

```
session_start();
session_unset();
session_destroy();
$_SESSION = array();
```

Registrazione di nuovi utenti

E se volessimo aggiungere nuovi utenti al database per effettuare un login? Innanzitutto, facciamo i dovuti controlli:

```
if(!isset($_POST["<nuovoUtenteInserito>"]) || !isset($_POST["<nuovaPasswordInserita>"]))
```

Se l'if restituisce true, mostriamo all'utente un form per l'inserimento dei dati relativi a una registrazione (in questo caso utente e password).

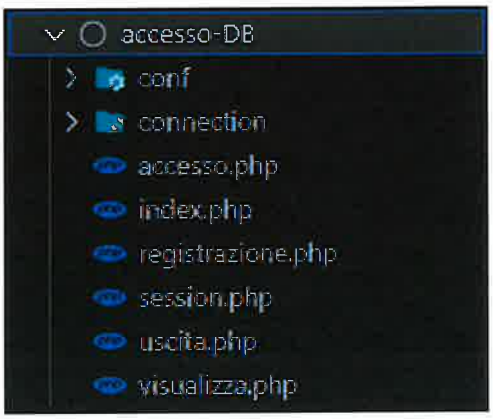
Altrimenti, scriviamo lo script per effettuare la registrazione vera e propria di un nuovo utente.

```
$username = $_POST["<nuovoUtenteInserito>"];
$password = $_POST["<nuovaPasswordInserita>"];
if(strlen($username) != 0 && strlen($password) != 0) {
    $passwordCrittografata = password_hash($password, PASSWORD_DEFAULT);
    require_once("path del file per l'apertura della connessione");
    $query = "SELECT * FROM <nomeTabella> WHERE <nomeColonnaUtenti> = ?";
    $stmt = $connection->prepare($query);
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $result = $stmt->get_result();
    if($result->num_rows != 0){
        echo "L'utente $username &egrave; gi&agrave; presente nel database.";
    } else {
        $query = "INSERT INTO <nomeTabella> (<nomeColonnaUtenti>, <nomeColonnaPassword>)
                VALUES ('$username', '$password')";
    if(!$connection->query($query)) {
        echo("Error description: " . $connection->error);
        echo "<br><br><a href='path del file per la pagina principale'>Home Page</a><br>";
    } else {
        echo " L'utente $username &egrave; stato aggiunto al database.";
```

```
            echo "<br><br><a href='path del file per la pagina principale'>Home Page</a><br>";
        }
            }
            $result->free();
            require_once("path del file per la chiusura della connessione");
        }
        else
            echo "Username/password non validi.";
```

Gestione delle sessioni

Aggiungiamo un file *session.php* al progetto.




```
<?php
    session_start();
    if(!isset($_SESSION["start_time"])){
        header("Location:<path del file per il login>");
        die();
    }
    else {
        $now = time();
        $time = $now - $_SESSION["start_time"];
        if ($time > <tempoInSecondi>) {
            header("Location:<path del file per il logout>");
            die();
        }
    }
?>
```

Tecniche di validazione in PHP

4.1. Validazione con funzioni native

- isset() e empty() per verifica esistenza/valore
- is_numeric(), is_string(), is_array(), is_bool()
- strlen() per controllare lunghezza
- ctype_*() per controlli su caratteri (alfabetici, numerici, ecc.)

4.2. Validazione con filter_var()

Utilizza filtri predefiniti:

- FILTER_VALIDATE_EMAIL
- FILTER_VALIDATE_URL
- FILTER_VALIDATE_INT
- FILTER_VALIDATE_BOOLEAN

Supporta anche opzioni e range tramite filter_var($value, $filter, $options)

4.3. Validazione con espressioni regolari

Utilizzo di preg_match() per pattern complessi (es. codici fiscali, password, numeri di telefono)

4.4. Validazione personalizzata

Definizione di funzioni ad hoc per regole specifiche non coperte dai metodi standard.

---

5. Gestione centralizzata della validazione

È buona pratica implementare un sistema modulare:

- Funzioni riutilizzabili (es. validateEmail($email))
- Classi di validazione (Validator, FormValidator)
- Array associativi per memorizzare errori ($errors['campo'] = 'messaggio')
- Separazione tra logica di validazione e logica di business

---

6. Validazione e sicurezza

6.1. Protezione contro XSS

- Usare htmlspecialchars() prima dell'output
- Sanitizzare i dati prima dell'uso in pagine HTML

6.2. Protezione contro SQL Injection

- Mai interpolare direttamente input utente nelle query
- Usare **prepared statements** con PDO o MySQLi

6.3. Validazione di file upload

- Controllare estensione e tipo MIME
- Limitare la dimensione
- Utilizzare finfo_file() per verifica reale del contenuto

ESPRESSIONI REGOLARI (REGEX)

Le **espressioni regolari**, comunemente note come "**regex**", sono stringhe di testo appositamente formattate utilizzate per trovare corrispondenze (match) all'interno di una stringa di testo oppure sostituire uno o più caratteri all'interno di una stringa.

Un'espressione regolare è formata da un pattern che contiene al suo interno una serie di simboli attraverso i quali è possibile identificare gruppi di stringhe.

Questa tabella mostra tutti i parametri che possono essere inseriti nel pattern:

Carattere	Descrizione
\	Carattere generico di escape
^	Delimitatore di inizio della stringa
$	Delimitatore di fine della stringa
.	Tutto eccetto il carattere di invio
[	Carattere di inizio della definizione di classe
]	Carattere di fine della definizione di classe
|	Inizio di un ramo alternativo
(	Inizio subpattern
)	Fine subpattern
?	0 o 1 occorrenze
*	0 o più occorrenze
+	1 o più occorrenze
{	Inizio intervallo minimo/massimo di occorrenze
}	Fine intervallo minimo/massimo di occorrenze
–	Un range di caratteri all'interno di parentesi quadre
.	Un singolo carattere
\s	Un carattere di spaziatura (space, tab, newline)
\S	Tutto eccetto un carattere di spaziatura
\d	Un carattere numerico (0-9)
\D	Tutto eccetto un carattere numerico
\w	Una lettera (a-z, A-Z, 0-9, _)
\W	Tutto eccetto una lettera
[aeiou]	Uno dei caratteri compresi nella parentesi
[^aeiou]	Tutto eccetto i caratteri compresi nella parentesi
(abc|def|ghi)	Una delle alternative tra parentesi

Nota che un pattern, per essere interpretato come tale dal linguaggio PHP, deve iniziare con "**/^**" e terminare con "**$/**".

FUNZIONI PER LE ESPRESSIONI REGOLARI IN PHP

preg_match($pattern, $input, $matches): Verifica se viene trovata almeno una corrispondenza nella stringa **$input** in base all'espressione regolare **$pattern**. Tale numero viene ritornato in **$matches**.

preg_match_all($pattern, $input, $matches): Ricerca tutte le espressioni regolari **$pattern** all'interno della stringa **$input**. Le occorrenze trovate vengono ritornate in **$matches.**

preg_replace($pattern, $replacement, $input, $limit): Sostituisce tutte le occorrenze trovate nella stringa **$input** con la stringa **$replacement** in base all'espressione regolare **$pattern**. Se presente, il parametro **$limit** limita il numero di sostituzioni da effettuare.

preg_split($pattern, $input): Suddivide la stringa **$input** in sottostringhe utilizzando l'espressione regolare **$pattern**.

Nota: La funzione preg_match() interrompe la ricerca dopo aver trovato la prima occorrenza, mentre preg_match_all() continua la ricerca fino alla fine della stringa e trova tutte le possibili occorrenze invece di fermarsi alla prima.

USO DELLE REGEX NEL COSTRUTTO CONDIZIONALE IF

La funzione preg_match(), se inserita all'interno delle parentesi di un costrutto if(), restituisce il valore booleano **true** se la stringa in input alla funzione genera un match con il pattern.

```
if (preg_match($pattern, $stringa))
{
    // codice se la stringa genera un match con il pattern
}
else
{
    // codice se la stringa NON genera un match con il pattern
}
```

FILE UPLOAD

Nome file proveniente da pagina html (<input type="file" name="fileToUpload" id="fileToUpload">)

```
<?php
// Directory dove i file verranno salvati
$storage_dir = "<percorso sicuro>";


// Controlla se il form è stato inviato
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_FILES["fileToUpload"])) {
   $target_file = $storage_dir . basename($_FILES["fileToUpload"]["name"]);
   $uploadOk = 1;
   $fileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));



   // Controlla se il file esiste già
   if (file_exists($target_file)) {
      echo "Mi dispiace, il file esiste già";
      $uploadOk = 0;
   }

   // Controlla la dimensione del file ...massimo in Byte
   if ($_FILES["fileToUpload"]["size"] > <dimensione in byte>) {
      echo "Il tuo file è troppo grande";
      $uploadOk = 0;
   }

   // Limita i tipi di file permessi
   if($fileType != "<estensione1>" && $fileType != "<estensione2>" ) {
      echo "Solo <estensione1> e <estensione2> sono permessi.";
      $uploadOk = 0;
```

```
    }

    // Verifica se $uploadOk è impostato a 0 da un errore
    if ($uploadOk == 0) {
        echo "Mi dispiace, file non caricato.";
    } else {
        // Se tutto è ok, prova a caricare il file
        if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
            echo "Il file ". htmlspecialchars( basename( $_FILES["fileToUpload"]["name"])) . " è stato caricato";
//In PHP, la funzione basename() serve per estrarre il nome del file o dell'ultima parte di un percorso (path).
        } else {
            echo "Mi dispiace, file non caricato.";
        }
    }
}
?>
```

MANUALE DI SISTEMI

Sistemi e Reti

Protocolli e classi

Protocolli		
Protocollo	Porta	TCP/UDP
FTP-data	20	TCP
FTP	21	TCP
TELNET	23	TCP
SMTP	25	TCP
DNS	53	TCP/UDP
DHCP (Server)	67	UDP
DHCP (Client)	68	UDP
TFTP	69	UDP
HTTP	80	TCP
HTTPS	443	TCP
POP3 (e.mail)	110	TCP
MySQL	3306	TCP
SSL	465	TCP
SMTPS	465	TCP
IKE	500	UDP
RIP	520	UDP
TLS	587	TCP
FTPS (dati)	989	TCP
FTPS (comandi)	990	TCP
POPS	995	TCP

Indirizzi IP Classful		
Classe	Maschera di rete	Intervallo
A	255.0.0.0	0.0.0.0 - 127.255.255.255
B	255.255.0.0	128.0.0.0 - 191.255.555.255
C	255.255.255.0	192.0.0.0 - 223.255.255.255

Indirizzi di rete privati

Classe	Intervallo
A	10.0.0.0
B	172.16.0.0 - 172.31.0.0
C	192.168.0.0 - 192.168.255.0

Numero di porta

WELL KNOWN	0 - 1023
REGISTRATE	1024 - 49151
DINAMICHE	49152 - 65535

Comandi configurazione firewall

Configurare una ACL standard su un router:

- Router> enable
- Router# configure terminal
- Router(config)# access-list [numero identificativo] [azione] [IP mittente] [wildcard]
- Router(config)# interface [nome interfaccia]
- Router(config-if)# ip access-group [numero identificativo] [verso di applicazione]
- Router(config-if)# end
- Router# disable
- Router> ...

Per disassociare una ACL precedentemente assegnata (entrare in modalità configurazione sull'interfaccia designata) anteporre no al comando già visto:

Router(config-if)# no ip access-group [numero identificativo] [verso di applicazione]

Per eliminare una ACL (entrare in modalità configurazione), anteporre no al comando:

Router(config-if)# no access-list [numero identificativo]

Non si può eliminare una sola regola dell'ACL.

Parametri:

- Numero identificativo: compreso tra **1** e **99**.
- Azione: **permit** / **deny**.
- IP mittente: indirizzo IP del mittente, se specifico, oppure indirizzo di rete o metà rete.
- Wildcard: sequenza di 32 bit, che viene confrontata con l'IP mittente, ossia: dove i bit della wildcard hanno valore 0, i bit dell'IP mittente restano invariati, mentre dove la wildcard presenta bit a 1, l'IP mittente avrà i corrispondenti bit a 0.

Se la wildcard è 0.0.0.0, [IP mittente] [wildcard] possono essere sostituiti da **any**.

- Verso di applicazione: **in** / **out**.

Configurare una ACL estesa su un router:

- Router> enable
- Router# configure terminal
- Router(config)# access-list [numero identificativo] [azione] [protocollo usato] [IP mittente] [wildcard] [IP destinatario] [wildcard] {operatore} {numero di porta}
- Router(config)# interface [nome interfaccia]
- Router(config-if)# ip access-group [numero identificativo] [verso di applicazione]
- Router(config-if)# end
- Router# disable
- Router> ...

Vedi pagina precedente per eliminare o disassociare una ACL.

Router(config-if)# no ip access-group [numero identificativo] [verso di applicazione]

Router(config-if)# no access-list [numero identificativo]

- Numero identificativo: compreso tra **100** e **199**.
- Azione: **permit / deny**.
- IP mittente/destinatario: indirizzo IP del mittente/destinatario, se specifico, oppure indirizzo di rete o metà rete.
- Wildcard: sequenza di 32 bit, che viene confrontata con l'IP mittente/destinatario, ossia: dove i bit della wildcard hanno valore 0, i bit dell'IP restano invariati, mentre dove la wildcard presenta bit a 1, l'IP avrà i corrispondenti bit a 0.

Se la wildcard è 0.0.0.0, [IP mittente] [wildcard] o [IP destinatario] [wildcard] possono essere sostituiti (anche entrambi) da **any**.

- Operatore {opzionale} - correlato al numero di porta: **eq** = '=', **gt** = '>', **lt** = '<', **neg** = '≠'.
- Numero porta {opzionale}: specifica il numero di porta del processo su cui è in esecuzione.
- Verso di applicazione: **in / out**.

Bibliografia

- Manuale ufficiale di MySql: https://dev.mysql.com/doc/refman/8.0/en/ .
- Manuale ufficiale di PHP: https://www.php.net/manual/en/ .