Tipizzazione

La tipizzazione e' la definizione del tipo delle variabili come ad esempio `int` e `string`.

Mentre creiamo un programma PHP, che sia legato a SQL o no, non dobbiamo preoccuparci di questa, dato che PHP non e' **strongly typed.**

Questo vuol dire che non serve definire un tipo per le variabili; un linguaggio **loosely typed** generalmente e' piu' veloce e versatile dato che si possono velocemente creare e sostituire variabili, tuttavia e' molto meno chiaro e facilmente comprensibile della sua controparte.

Nelle ultime versioni di PHP si possono comunque definire dei tipi per gli parametri e i return delle funzioni.

Eccezioni

PHP e' simile agli altri linguaggi che abbiamo usato. Per gestirele eccezioni si puo' usare `throw` o `try & catch & finally` .

Throw

```
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}
```

La keyword `throw` lancia un exception e tutto il codice seguente non verra' eseguito; se questa non viene fermata attraverso un `catch` il programma terminera' con un exception: ***"Uncaught Exception".***

Try, Catch, Finally

```php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide. ";
} finally {
  echo "Process complete.";
}
```

Conosciamo gia `try & catch` quindi passiamo subito a `finally`. il codice all' interno di questa keyword viene eseguito sempre, a prescindere dal fatto che l'*exception* venga presa con `catch` o meno.

Metodi per le eccezioni: devdocs.io/php/class.exception
Credits: w3schools.com/php/php_exceptions

Query e gestione dati

Ci sono diversi modi per mandare e interagire con le query in PSQL con PHP:

Query

pg_query

```
$conn = pg_pconnect("dbname=publisher");
if (!$conn) {
  echo "An error occurred.\n";
  exit;
}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
  echo "An error occurred.\n";
  exit;
}

while ($row = pg_fetch_row($result)) {
  echo "Author: $row[0]  E-mail: $row[1]";
  echo "<br />\n";
}
```

Con `pg_query()` la *query* viene mandata istantaneamente a differenza di altry metodi.
Se avviene un errore e il return di `pg_query()` e' `false` si puo' vedere i dettagli di questo con `pg_last_error()`.

pg_query_params

```
$dbconn = pg_connect("dbname=mary");

$result = pg_query_params($dbconn, 'SELECT * FROM $2 WHERE name =
$1', array("Joe's Widgets",”shops”));

// Comparazione a una pg_query “normale”
$str = pg_escape_string("Joe's Widgets");
$result = pg_query($dbconn, "SELECT * FROM shops WHERE name =
'{$str}'");
```

La differenza con `pg_query()` e' che si possono **definire separatamente le variabili PHP da inserire nella query**; questo avviene passando un array delle variabili come parametro e definendo nella stringa le posizione delle variabili attraverso i codici corrispettivi `$1, $2, …` , questi possono essere utilizzati piu' volte nella stringa.
Perche' e' meglio di `pg_query()` ? Perche' quando si mandano delle query spesso si va incontro a errori causati dai **caratteri di escape** come `\n, \", \\` e usando questa funzione si cerca di evitare questi casi definendo separatamente le variabili.

Gestione dati

Per la gestione dei dati di una risposta PSQL dobbiamo preoccuparci principalmente di come ricavarne i dati effettivi e sotto che forma.
Ci sono molti metodi per interagirci quindi li andremo a vedere i piu' importanti direttamente sulla documentazione.

Ripasso veloce librerie o array associativi

```php
// Creazione
$a_array = [
        "Key" => "Value",
        1 => "Stringa",
        "Integer" => 1,
    ];

// Iterazione e recupero dati
foreach($a_array as $key => $value){
    echo $key." ".$value."<br>";
}

// Aggiunta valori
$a_array["Nuovo"] = "Valore";

// Esecuzione funzione e ricerca
array_search("Stringa", $a_array); // => 1
```

Esempio di memorizzazione e uso dati:

```
$query = "SELECT * FROM forniture";
$query_r = pg_query($conn, $query);
$result = [];
for ($i=0; $i < pg_num_rows($query_r); $i++) {
    //Metodo di salvataggio dati non consigliato in quanto con
    //un database molto grande potrebbe portare a problemi di
memoria
    array_push($result, pg_fetch_assoc($query_r));
}

echo "<table border='1'>";
echo "<tr><td>f_cod</td><td>a_cod</td><td>quantita</td></tr>";
foreach ($result as $a) {
    echo "<tr>";
    foreach ($a as $key => $value) {
        echo "<td>".$value."</td>";
    }
    echo "</tr>";
}
echo "</table>";
```

Funzioni piu' importanti:

- *pg_fetch_assoc()*
- *pg_fetch_array()*
- *pg_fetch_all()*
- *pg_num_rows()*
- *pg_affected_rows()*

- *pg-convert()*
- *pg-insert()*
- *pg-update()*
- *pg-delete()*

Credits: devdocs.io/php-database-postgresql

DCL

Data **C**ontrol **L**anguage, e' il linguaggio di SQL usato per gestire utenti e privilegi a loro assegnati.

Sintassi base e opzioni

```
[GRANT | REVOKE]
privilege : { USAGE | SELECT | INSERT | UPDATE | DELETE | TRUNCATE
| DROP | ALTER | EXECUTE | ALL }
ON
object : { ALL TABLES | TABLE table_name | ALL DATABASES |
DATABASE database_name | ALL SCHEMAS | SCHEMA schema_name | ALL
FUNCTIONS | FUNCTION function_name() }
TO
'username'@'address';

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::;

ALTER USER
'username'@'address'
WITH
option : { SUPERUSER | CREATEDB | CREATEROLE };
```

Attraverso `GRANT, REVOKE` **possiamo attribuire a i vari utenti vari permessi di controllo sul database**. Questo e' ovviamente utile in un contesto reale dove il DB e' consultato da piu' persone/utenti e non tutte possono e devono avere gli stessi permessi, come per esempio quello di eseguire un `DROP` sul database.
`ALTER USER` invece permette di modificare alcuni settaggi dell'utente, impostati quando quet'ultimo e' stato creato; **questi permessi sono di grande importanza come creare database o essere superuser.**

Promemoria: `ROLE` e' un alias per `USER` , per questo molte volte si vedono interscambiati

Creazione di un utente

```
CREATE USER
user_name
WITH
option : {   SUPERUSER | NOSUPERUSER
           | CREATEDB | NOCREATEDB
           | CREATEROLE | NOCREATEROLE
           | INHERIT | NOINHERIT
           | LOGIN | NOLOGIN
           | REPLICATION | NOREPLICATION
           | BYPASSRLS | NOBYPASSRLS
           | CONNECTION LIMIT connlimit
           | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
           | VALID UNTIL 'timestamp'
           | IN ROLE role_name [, ...]
           | IN GROUP role_name [, ...]
           | ROLE role_name [, ...]
           | ADMIN role_name [, ...]
           | USER role_name [, ...]
           | SYSID uid
     }
```

Credits: geeksforgeeks.org/mysql-grant-revoke-privileges
Credits: onecompiler.com/tutorials/postgresql/commands/dcl-commands
Credits: phoenixnap.com/kb/postgres-create-user

Esercizio

- Connettiti a un database, nel caso in cui fallisca usa la gestione delle exception per segnalare l'errore.
- Memorizza tutti i dati di una tabella in un array associativo attraverso una query e stampali in una tabella.
- Attraverso un' altra query crea un nuovo user e dagli il permesso `SELECT`, dopodiche disconnettiti dal server.
- Connettiti come il nuovo user, richiedi e ristampa la tabella

---

Link utili

- **W3School** : *W3School - PHP*
- **DevDocs** : *devdocs.io/php-database-postgresql*