Diario di lavoro

Titolo progetto	Gestione ore
Luogo	Trevano-Canobbio
Data	15 ottobre 2019

Lavori svolti

Oggi ho proseguito con l'implementazione delle classi Model che si interfacciano al database ed eseguono le query su di esso.
Più di preciso sono riuscito a terminare la funzione 'addActivity' creata la scorsa giornata di lavoro. Questa funzione si occupa di aggiungere una riga all'interno della tabella 'lavoro' del database. Il risultato è il seguente:

```
/**
     * Inserts a new row into table 'lavoro' of database with data passed as parameter.
     * @param Activity $activity The data to add to the database.
     * @return bool true if the insert operation is successful, false otherwise.
     */
    public function addActivity(Activity $activity): bool
    {
        //Save starting and delivery dates into variables
        $startDate = $activity->startDate;
        $deliveryDate = $activity->deliveryDate;
        //Convert starting and delivery dates to strings
        $startDateString = $startDate->format("Y-m-d");
        $deliveryDateString = $deliveryDate->format("Y-m-d");
        //Write the query that will write into the database.
        $query = "INSERT INTO lavoro(nome, note, data_inizio, data_consegna, ore_preven
tivate) VALUES (:name, :notes, :startDate, :deliveryDate, :hours)";
        //Prepare the query.
        $statement = $this->database->prepare($query);
        //Replace placeholder ':name' with activity name taken from field 'name' of 'ac
tivity' parameter.
        $statement->bindParam(":name", $activity->name);
        //Replace placeholder ':notes' with activity notes taken from field 'notes' of
'activity' parameter.
        $statement->bindParam(":notes", $activity->notes);
        //Replace placeholder ':startDate' with activity starting
        //date taken from field 'startDate' of 'activity' parameter.
        $startDate = $activity->startDate;
        $statement->bindParam(":startDate", $startDateString);
        //Replace placeholder ':deliveryDate' with activity delivery
        //date taken from field 'deliveryDate' of 'activity' parameter.
        $statement->bindParam(":deliveryDate", $deliveryDateString);
        //Replace placeholder ':hours' with activity estimated hours
        //taken from field 'estimatedHours' of 'activity' parameter.
        $statement->bindParam(":hours", $activity->estimatedHours);
```

```
        //Return the result of query execution
        $result = $statement->execute();
        return $result;
    }
```

In seguito mi sono occupato della prossima funzione da creare: “eliminaLavoro”, tradotta in “deleteActivity”. Questa funzione cancella una riga dalla tabella ‘lavoro’ del database basandosi sulla sua chiave primaria. È composta dal codice sottostante, che fa utilizzo della funzione “deleteModel” ereditata dalla superclasse Model (vedi Problemi riscontrati e soluzioni adottate):

```
    /**
     * Deletes a record from the MySQL table 'lavoro' where the name is equal to the na
me of an object of type Activity.
     * @param Activity $activity The data of the activity to delete.
     * @return bool true if the deletion is successful, false otherwise.
     */
    public function deleteActivity(Activity $activity): bool
    {
        return $this->deleteModel([$activity->name]);
    }
```

Problemi riscontrati e soluzioni adottate

Quando mi sono trovato a verificare il funzionamento della funzione ‘addActivity’ creata la scorsa giornata di lavoro, inizialmente dava il seguente messaggio di errore:

```
Recoverable fatal error: Object of class DateTime could not be converted to string
```

Questo perché la funzione inseriva nel database direttamente il valore del campo ‘startDate’ del parametro passatogli, di tipo DateTime:

```
$statement->bindParam(":startDate", $activity->startDate);
```

Ovviamente per la query, che si aspetta un tipo string per tutti i valori che le vengono assegnati, non andava bene, quindi ho cambiato il codice sopracitato in:

```
$statement->bindParam(":startDate", $activity->startDate->format("Y-m-d"));
```

Ma qui il messaggio di errore cambiava nel seguente:

```
Notice: Only variables should be passed by reference
```

Questo perché pare che a PHP dia fastidio richiamare un metodo tramite la proprietà di un oggetto, che è esattamente quello che sto facendo io, quindi ho dapprima cambiato nuovamente l’istruzione in:

```
$startDate = $activity->startDate;
$statement->bindParam(":startDate", $startDate->format("Y-m-d"));
```

Ma l’errore rimaneva, allora ho sostituito con:

```
$startDate = $activity->startDate;
$startDateString = $startDate->format("Y-m-d");
$statement->bindParam(":startDate", $startDateString);
```

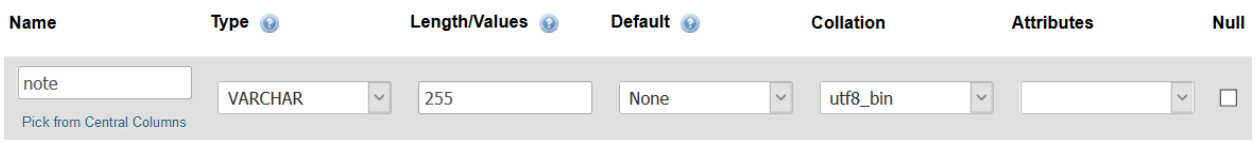
E questo ha finalmente fatto tacere l’errore, ma guardando la tabella tramite PHPMyAdmin ho constatato che l’insert non veniva comunque eseguito. Ho quindi stampato le informazioni di errore con il seguente codice. Prima era:

```
//Return the result of query execution
return $statement->execute();
```

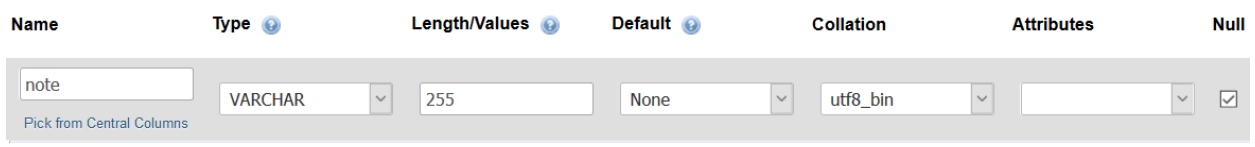
Adesso l’ho temporaneamente fatto diventare:

```
//Return the result of query execution
$result = $statement->execute();
var_dump($statement->errorInfo());
return $result;
```

Questo mi ha permesso di leggere che l'errore era causato da un errore di implementazione del database: non ho impostato che il campo 'note' della tabella 'lavoro' potesse anche essere nullo, e la riga di esempio che andavo a inserire avevo proprio questo valore per il campo 'note'.
Ho sistemato quindi tramite PHPMyAdmin.
Prima:



Dopo:

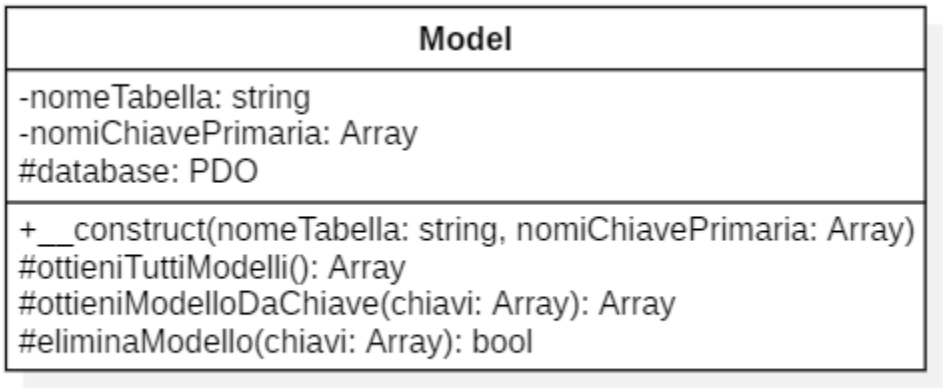


Dopo tutto questo lavoro ho ripensato al fatto che la funzione 'getActivityByName' contiene molto codice che potrebbe essere condiviso con le altre classi Model, quindi l'ho generalizzata e messa nella superclasse con il nome di 'getModelByKey' in modo da dover scrivere meno codice per le altre sottoclassi. Questa funzione contiene il seguente codice:

```
    /**
     * Gets a single model identified by its primary key.
     * @param array $keys The primary key values for a row.
     * @return array The results of the select query as a bidimensional array containin
g one or zero rows of the table
     * with name tableName.
     */
    protected function getModelByKey(array $keys): array
    {
        //Write the beginning of the query to search in the database
        $query = "SELECT * FROM $this->tableName WHERE ";
        //Compose the query using
        for ($i = 0; $i < count($keys) - 1; ++$i) {

            $query .= $this->primaryKeyNames[$i] . " = :name$i AND ";

        }

        $query .= $this-
>primaryKeyNames[count($keys) - 1] . " = :name" . (count($keys) - 1);
        //Prepare the query
        $statement = $this->database->prepare($query);
        //Assign to placeholders ':nameN' the value of parameter 'keys'
        for ($i = 0; $i < count($keys); ++$i) {
            $statement->bindParam(":name$i", $keys[$i]);
        }

        //Execute the query
        $statement->execute();
        //Fetch the results
        $queryResult = $statement->fetchAll();
        //Return the results
```

```
        return $queryResult;
    }
```

Questo ha richiesto una modifica della progettazione della classe Model, che ho aggiornato e adesso ha quest'aspetto:



È possibile notare l'aggiunta del campo 'nomiChiavePrimaria' e delle due funzioni "ottieniModelloDaChiave" e "eliminaModello": il campo '*nomiChiavePrimaria*' è un array contenente uno o più valori che corrispondono al nome del campo impostato come chiave primaria o, in caso di una chiave composta, ai nomi dei campi impostati come chiave.
La funzione "*ottieniModelloDaChiave*" esegue una query al database e ritorna un array contenente un array associativo in cui ogni elemento corrisponde a un campo del database associato al rispettivo valore della riga della tabella la cui chiave corrisponde a quella o quelle specificate come parametro.
Infine, la funzione "*eliminaModello*" esegue una query al database per cancellare una riga la cui chiave corrisponde a quella o quelle passate come parametro dalla tabella il cui nome è specificato nel campo nomeTabella.
In seguito mi sono accorto che il codice per comporre la condizione della query era uguale sia per la funzione "ottieniModelloDaChiave" che per "eliminaModello", quindi l'ho generalizzato in una funzione chiamata "composePrimaryKeyCondition", visibile di seguito:

```
    /**
     * Composes the part of a MySQL query that contains the conditions where the primar
y key is equal to a value.
     * @param array $keys The names of the fields that make up the primary key.
     * @return string The condition part of a MySQL query that checks if the primary ke
y is equal to a value.
     * For example:
     * composePrimaryKeyCondition(["primary_key_1", "primary_key_2"]);
     * Will return: "primary_key_1 = :key1 AND primary_key_2 = :key2".
     */
    private function composePrimaryKeyCondition(array $keys): string
    {
        $condition = "";
        for ($i = 0; $i < count($keys) - 1; ++$i) {
            $condition .= $this->primaryKeyNames[$i] . " = :key$i AND ";
        }
        $condition .= $this-
>primaryKeyNames[count($keys) - 1] . " = :key" . (count($keys) - 1);
        return $condition;
    }
```

Alla fine sono riuscito a implementare anche questa nuova funzione “eliminaModello”, che funziona grazie al codice sorgente visibile più in basso:

```php
    /**
     * Deletes a record from the MySQL table whose name is equal to the value of field 'tableName'.
     * @param array $keys The table's primary key values for a row.
     * @return bool true if the deletion is successful, false otherwise.
     */
    protected function deleteModel(array $keys): bool
    {
        //Write the beginning of the query to delete from a table in the database
        $query = "DELETE FROM $this->tableName WHERE ";
        //Compose the condition of the query
        $query .= $this->composePrimaryKeyCondition($keys);
        //Prepare the query
        $statement = $this->database->prepare($query);
        //Assign to placeholders ':nameN' the value of parameter 'keys'
        for ($i = 0; $i < count($keys); ++$i) {
            $statement->bindParam(":key$i", $keys[$i]);
        }
        //Execute the query
        return $statement->execute();
    }
```

Punto della situazione rispetto alla pianificazione
Sono in anticipo rispetto alla pianificazione.

Programma di massima per la prossima giornata di lavoro
Portare avanti la documentazione della pianificazione e la realizzazione delle classi Model che si interfacciano al database.