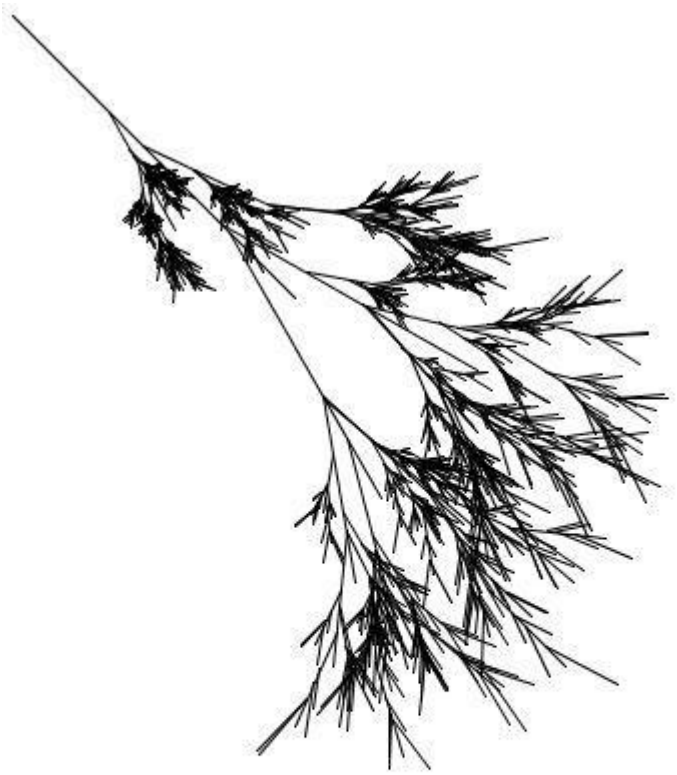


```php
<?php
    echo “programmeren in php”;
    echo “<br>”;
    echo “php en de database”;
?>
```

info.broklede@gmail.com

Voorblad:
“Plant” getekend met de programmeertaal Processing, gebruik makend van recursie.

Inhoud

1

PHP en de database

1.1 Contact met de database

Eén van de meest voorkomende toepassingen van PHP op een webserver is die waarbij PHP contact maakt met een database. Omdat we al gewend zijn aan het werken met WebServer en SQLite (of een soortgelijk pakket) zullen we daar nu ook mee aan de slag gaan.
En die SQLite database gaan we benaderen met behulp van PHP met de zogenaamde **PDO extensie**.

PDO is de afkorting van **P**HP **D**ata **O**bjects. Je hebt al eerder met objecten binnen PHP gewerkt en de PHP Data Objects zijn daar een uitbreiding op. Het is gemaakt om diverse databases te kunnen benaderen zonder al teveel aan de PHP-code te moeten veranderen. Het is als het ware wat extra software tussen PHP en de database. Laten we eerst de belangrijkste mogelijkheden ervan bekijken.

Het eerste wat je moet doen om vanuit PHP met een database te kunnen werken is de verbinding naar die database tot stand te brengen. Gelukkig is dat niet zo heel erg moeilijk. Dat gaat als volgt in een stukje PHP-code:

```php
<?php
  // Maken van verbinding
  $db = new PDO("sqlite: ..hier de naam van het SQLite bestand..");

  // De volgende regel is bedoeld om wat duidelijker foutmeldingen
  // te krijgen
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // Dan komt hier de code waarin je gebruik maakt van de database
?>
```

Wat je in feite hier doet is een nieuwe (new) instantie (object) van de PDO klasse maken. Bij het aanmaken ervan wordt met
`("sqlite: ..hier de naam van het SQLite bestand..")`
een aantal parameters meegegeven.

Deze parameters kunnen verschillen per database. Maak je bijvoorbeeld gebruik van een MySQL database dan gebruik je de bijvoorbeeld de volgende parameters:
`("mysql:host=localhost;dbname=test","user", "password")`

Maar SQLite is lekker eenvoudig, geen **gebruikersnaam**, geen **wachtwoord** en het draait ook niet als een server dus ook **localhost** is overbodig.

Het nieuwe object wordt dan aangemaakt en er wordt in de objectvariabele `$db` een verwijzing naar dat object opgeslagen.

De verbinding met de database kun je weer sluiten door met `$db = NULL` de verwijzing in de objectvariabele `$db` gelijk te stellen aan `NULL` waarna het object niet meer bestaat.
Laten we het eens met een echte database proberen en kijken wat we er dan mee kunnen.

Om goed te oefenen kunnen we natuurlijk een groot oefenbestand aanmaken. Handiger is het om een bestand te maken door gebruik te maken van een reeks SQL opdrachten die in een bestand zitten. Kijk maar in **de lessen over SQL** hoe we dat eerder gedaan hebben.

Opdracht I

- Maak het bestand winkel.sqlite aan (als je dat nog niet hebt) zoals beschreven in de lessen over SQL in de opdrachten 2 en 3

Voor we aan de slag gaan is het goed nog even te kijken naar de structuur van de tabellen die we nu in de database winkel hebben. Want om goed te kunnen oefenen moet je de velden kennen.

In het bestand **winkel.sqlite** staan nu de volgende drie tabellen:

Artikelen	Bestellingen	Klanten
artikelnummer	bestellingnummer	klantnummer
artikelnaam	klantnummer	voornaam
prijs	artikelnummer	achternaam
	order_datum	woonplaats
	hoeveelheid	provincie

In de `SELECT` opdrachten zullen we gebruik maken van de namen van de velden.

Nu we de beschikking hebben over een echte database kunnen we aan de slag met het benaderen van die database met behulp van PHP.

1.2 Gegevens lezen

Opdracht 2

- Zorg dat WebServer gestart is
- Typ de onderstaande code in jouw editor
- Sla het bestand op in de rootmap (www, htdocs of root) onder de naam **opdracht2.php**
- Start daarna het programma door in de browser te typen: **http://localhost/opdracht2.php**

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>De klanten van de winkel</title>
  </head>
  <body>
    <?php
      // Maken van verbinding
      $db = new PDO("sqlite:../data/winkel.sqlite");

      // De code hieronder is om eventuele foutmeldingen
      // weer te geven
      $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      // Dan komt hier de code waarin je gebruik maakt van
      // de database
      $sql = "SELECT achternaam FROM klanten";
      $resultaat = $db->query($sql);

      foreach($resultaat as $row) {
        echo $row['achternaam'].'<br>';
      }
```

```
    // Sluiten van verbinding
    $db = NULL;
  ?>
 </body>
</html>
```

En? Zie je het lijstje achternamen uit de klantentabel? Dat is namelijk de bedoeling. Mocht je de lijst niet zien dan ben je misschien vergeten WebServer te starten? Of misschien heb jij de inlogcode van phpLiteAdmin gewijzigd? In dat geval zul je de code daarop aan moeten passen.

Eerst maar even wat uitleg van deze code.
De verbinding met de database wordt met de regel
`$db = new PDO("sqlite:../data/winkel.sqlite");`
tot stand gebracht.
Je ziet dat de database-naam `winkel.sqlite` hier in de code staat.

De verbinding is een object en wordt in de objectvariabele **`$db`** opgeslagen. **`$db`** noemen we ook wel de **database-handle**. Als we verderop in de code de database gaan benaderen dan doe we dat door gebruik te maken van de database-handle.

De regel `$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);` zorgt ervoor dat, als er iets mis is met de verbinding, we een melding op het scherm krijgen.
Wel handig want anders kun je uren zoeken naar de fout als de query niet lijkt te werken.

Met de regel: `$sql = "SELECT achternaam FROM klanten";` wordt de query samengesteld en in de variabele `$sql` opgeslagen.

De PDO klasse heeft twee methodes voor het uitvoeren van queries op de database, namelijk `query()` en `exec()`
`query()` is het eenvoudigst, die voert de query uit en geeft het resultaat terug.
`exec()` doet dat ook maar geeft naast het resultaat ook het aantal aangepaste rijen terug.
Hier gebruiken we eerst `query()`

De zin: `$resultaat = $db->query($sql);` zal nu ook wel duidelijk zijn ook al is de notatie misschien een beetje vreemd. Op deze manier wordt de query op de database (die wordt aangegeven met `$db`) uitgevoerd en het resultaat van de query wordt in `$resultaat` opgeslagen.

Tenslotte het stukje code waarmee het resultaat op de webpagina wordt weergegeven.

```
foreach($resultaat as $row) {
  echo $row['achternaam'].'<br>';
}
```

Hier wordt gebruik gemaakt van een **foreach-loop**.
Voor iedere rij in **`$resultaat`** wordt de achternaam naar het scherm ge'**`echo`**'d. Met de `<br>` erachter krijgen we ze keurig steeds op een nieuwe regel afgedrukt.
Herinner jij je nog dat je met een punt strings aan elkaar kunt koppelen? Voor HTML staat daardoor `<br>` meteen achter de achternaam.

Opdracht 3

- Maak op dezelfde wijze een PHP programma waarmee je de artikelnaam en de bijbehorende prijs uit de artikelen-tabel krijgt.

(Je zult merken dat de uitvoer mogelijk niet zo mooi is. De prijs komt vlak achter de artikelnaam. Vaak wordt een dergelijke uitvoer in een tabel gepresenteerd maar met behulp van JavaScript en CSS kan het nog veel mooier. Voorlopig is dat voor ons niet van belang.)

1.3 Gegevens toevoegen

Op soortgelijke manier kunnen we ook een klant toevoegen.

Opdracht 4

- Maak een PHP programma
- Plaats er de hieronder staande code in
- Sla het bestand op in de rootmap (www, htdocs of root) onder de naam **`opdracht4.php`**

- Start daarna het programma door in de browser te typen:
  `http://localhost/opdracht4.php`

```php
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Een klant toevoegen</title>
  </head>
  <body>
    <?php
      // Maken van verbinding
      $db = new PDO("sqlite:../data/winkel.sqlite");

      // De code hieronder is om eventuele foutmeldingen
      // weer te geven
      $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      // De klant toevoegen
      $sql = "INSERT INTO klanten (
                voornaam,
                achternaam,
                woonplaats,
                provincie
              )
              VALUES (
                'Karel',
                'Doorman',
                'Assen',
                'Drenthe'
              )";

      $resultaat = $db->exec($sql);

      echo '<p>Aantal toegevoegde klanten: '.$resultaat.'</p>';

      // De klant is toegevoegd maar welk klantnummer
      // heeft hij gekregen?
      // Dat gaan we eerst opvragen
      $laatste = $db->lastInsertId();

      // Aan de hand van het klantnummer kunnen we nu ook
      // ter controle de gegevens van de klant opvragen
      $sql = "SELECT * FROM klanten WHERE klantnummer = ".$laatste;
      $resultaat = $db->query($sql);

      // En we zetten ze even keurig in een tabel
      echo '<table>';
```

```
      foreach($resultaat as $row) {
        echo '<tr><td>Klantnummer</td><td>'.$row['klantnummer'].'</td></tr>';
        echo '<tr><td>Voornaam</td><td>'.$row['voornaam'].'</td></tr>';
        echo '<tr><td>Achternaam</td><td>'.$row['achternaam'].'</td></tr>';
        echo '<tr><td>Woonplaats</td><td>'.$row['woonplaats'].'</td></tr>';
        echo '<tr><td>Provincie</td><td>'.$row['provincie'].'</td></tr>';
      }
      echo '</table>';

      $db = NULL;
    ?>
  </body>
</html>
```

Is het je ook opgevallen dat het klantnummer ingevuld is terwijl dat helemaal niet in de code hierboven staat.
De klantentabel is destijds aangemaakt met de volgende code:

```
CREATE TABLE "klanten" (
    "klantnummer" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "voornaam" TEXT,
    "achternaam" TEXT,
    "woonplaats" TEXT,
    "provincie" TEXT
);
```

Achter klantnummer zie je een paar belangrijke attributen staan.

`PRIMARY KEY` betekent dat klantnummer het sleutelveld is.

`AUTOINCREMENT` betekent dat dit veld, iedere keer als er een nieuwe klant wordt toegevoegd, automatisch met 1 verhoogd wordt.

`NOT NULL` tenslotte betekent dat dit veld niet leeg gelaten mag worden. Ook wel logisch, anders heb je er niets aan als sleutelveld.

1.4 Gegevens wijzigen

Oeps, een foutje gemaakt.
De klant die we in moesten voeren heet niet Karel Doorman maar Karel Noorman.

Kortom, we moeten de gegevens wijzigen.
Aangezien het klantnummer uniek is, het is immers het sleutelveld, zullen we Karel Doorman aan de hand van het klantnummer op moeten zoeken en daarna wijzigen. In opdracht 5 gaan we ervan uit dat hij klantnummer 18 heeft. Uiteraard vul je een ander klantnummer in als hij bij jou een ander nummer heeft.

Opdracht 5

- Maak een PHP programma
- Plaats er de hieronder staande code in
- Sla het bestand op in de rootmap (www, htdocs of root) onder de naam **opdracht5.php**
- Start daarna het programma door in de browser te typen: **http://localhost/opdracht5.php**

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Een klant wijzigen</title>
  </head>
  <body>
    <?php
      // Maken van verbinding
      $db = new PDO("sqlite:../data/winkel.sqlite");

      // De code hieronder is om eventuele foutmeldingen
      // weer te geven
      $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      // De klant wijzigen
      $sql = "UPDATE klanten SET achternaam = 'Noorman' WHERE klantnummer = 18";

      $resultaat = $db->exec($sql);

      echo '<p>Aantal gewijzigde klanten: '.$resultaat.'</p>';

      // De klant is toegevoegd maar ter controle
      // willen we dat ook wel even zien
      $sql = "SELECT * FROM klanten WHERE klantnummer = 18";
      $resultaat = $db->query($sql);

      // En we zetten ze even keurig in een tabel
```

```
      echo '<table>';
      foreach($resultaat as $row) {
        echo '<tr><td>Klantnummer</td><td>'.$row['klantnummer'].'</td></tr>';
        echo '<tr><td>Voornaam</td><td>'.$row['voornaam'].'</td></tr>';
        echo '<tr><td>Achternaam</td><td>'.$row['achternaam'].'</td></tr>';
        echo '<tr><td>Woonplaats</td><td>'.$row['woonplaats'].'</td></tr>';
        echo '<tr><td>Provincie</td><td>'.$row['provincie'].'</td></tr>';
      }
      echo '</table>';

      $db = NULL;
    ?>
  </body>
</html>
```

De code van opdracht 4 en opdracht 5 zal verder niet zo moeilijk te begrijpen zijn. Het enige echte verschil is de regel die begint met **`$sql`**. Met **`INSERT INTO`** voeg je dus een record toe aan de tabel, met **`UPDATE`** kun je een record wijzigen. Bij **`UPDATE`** moet je er uiteraard wel goed voor waken dat je het juiste record wijzigt. Vandaar dat dat altijd moet gebeuren met behulp van het sleutelveld omdat een record daaraan uniek is te herkennen.

1.5 Gegevens verwijderen

Helaas is klant Karel Noorman zo boos geworden omdat we zijn naam verkeerd geschreven hebben dat hij uit het klantenbestand verwijderd wil worden.
Gelukkig is ook dat niet zo moeilijk.

Opdracht 6

- Maak een PHP programma
- Plaats er de hieronder staande code in
- Sla het bestand op in de rootmap (www, htdocs of root) onder de naam **`opdracht6.php`**
- Start daarna het programma door in de browser te typen: **`http://localhost/opdracht6.php`**

```php
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>Een klant verwijderen</title>
  </head>
  <body>
    <?php
      // Maken van verbinding
      $db = new PDO("sqlite:../data/winkel.sqlite");

      // De code hieronder is om eventuele foutmeldingen weer te geven
      $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      // Om te weten om welke klant het gaat zetten we
      // nog even de klantgegevens op het scherm

      $sql = "SELECT * FROM klanten WHERE klantnummer = 18";
      $resultaat = $db->query($sql);

      echo '<table>';
      foreach($resultaat as $row) {
        echo '<tr><td>Klantnummer</td><td>'.$row['klantnummer'].'</td></tr>';
        echo '<tr><td>Voornaam</td><td>'.$row['voornaam'].'</td></tr>';
        echo '<tr><td>Achternaam</td><td>'.$row['achternaam'].'</td></tr>';
        echo '<tr><td>Woonplaats</td><td>'.$row['woonplaats'].'</td></tr>';
        echo '<tr><td>Provoncie</td><td>'.$row['provincie'].'</td></tr>';
      }
      echo '</table>';

      // De klant verwijderen
      $sql = "DELETE FROM klanten WHERE klantnummer = 18";

      $resultaat = $db->exec($sql);

      // Een eenvoudige controle
      if ($resultaat == 1) {
        echo '<br>Deze klant is nu verwijderd.';
      }
      else {
        echo '<br>Deze klant is niet verwijderd.';
      }

      $db = NULL;
    ?>
  </body>
</html>
```

Ook in deze code staat niet veel nieuws.
Uiteraard horen de klantnummers in de `SELECT` regel en de `DELETE` regel aan elkaar gelijk te zijn.
Maar om het programma helemaal te kunnen testen zou je voor het klantnummer in de `SELECT` regel een bestaand klantnummer in kunnen vullen en in de `DELETE` regel een niet bestaand klantnummer.

2

CRUD

en de slimmere code

2.1 Wat is CRUD?

Wat we in hoofdstuk 1 hebben gedaan is het toepassen van de 4 onderdelen van **CRUD**. Dat staat namelijk voor **C**reate, **R**ead, **U**pdate en **D**elete. En dat zijn nu eenmaal de belangrijkste acties die we met een database ondernemen.

Voordat we echter een nieuwe CRUD toepassing gaan maken wordt het tijd om eens goed na te denken of het ook wat slimmer kan.

Een bekend principe bij het programmeren is **DRY** oftewel, **d**on't **r**epeat **y**ourself. Ook wel logisch. Als je wat moet veranderen, dat wil je dan toch maar op 1 plek doen!
En dat kan bijvoorbeeld met **object georiënteerd programmeren**. En door functies die we vaak moeten gebruiken, zoals het verbinden met de database, toch maar 1 keer te maken en dan in andere modules te gebruiken.

Maar laten we eerst eens gaan kijken wat we willen maken.

2.2 De speel-o-theek database

In een speel-o-theek kun je speelgoed lenen. Een soort van bibliotheek dus maar dan met speelgoed.

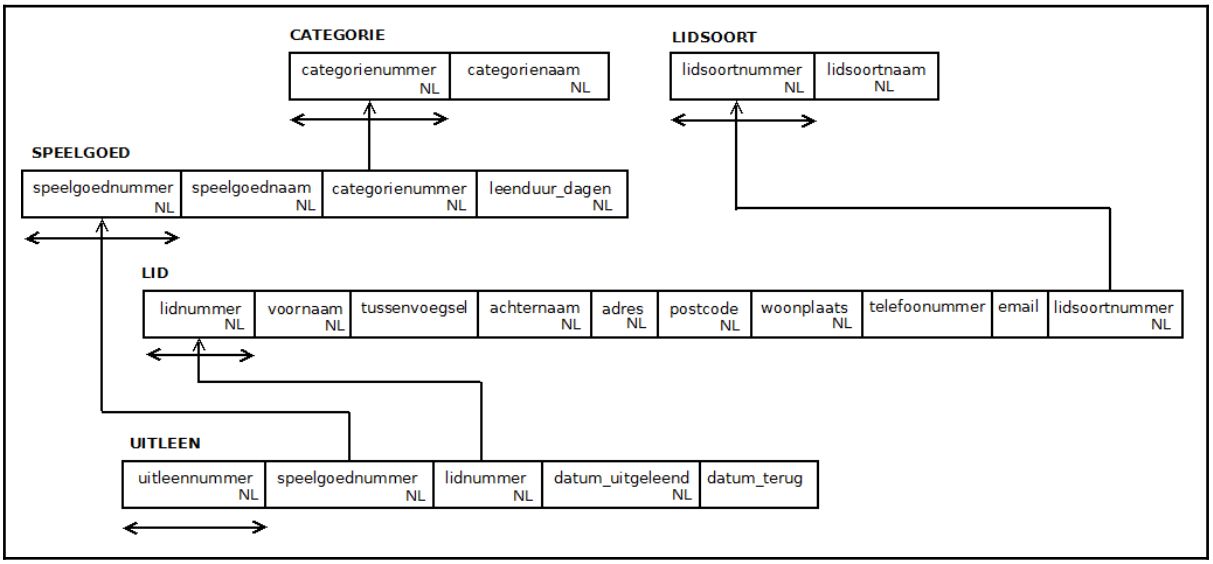
Een speel-o-theek heeft leden die het speelgoed mogen lenen.
Leden kun je indelen in lidsoorten.
Je zou bijvoorbeeld een indeling kunnen maken op basis van het aantal kinderen.
Immers de contributie kan afhankelijk zijn van het aantal kinderen.
Ook kun je op basis van het aantal kinderen meer speelgoed tegelijkertijd lenen.

Wat het speelgoed betreft, daarin heb je categorieën. Bijvoorbeeld:
Constructie (zoals K'nex, Lego), Fantasie materiaal (zoals piratenschip, Playmobil boerderij), Gezelschapsspellen (zoals Set, Kolonisten van Catan) en Puzzels.

Tenslotte heb je de uitleningen. Als een lid een stuk speelgoed leent wordt de

datum_uitgeleend ingevuld. Als dat speelgoed weer terug gebracht wordt, wordt de datum_terug ingevuld.
Dus het uitgeleende speelgoed vind je door in de uitleentabel te kijken van welk speelgoed de datum_terug nog leeg is.

Het strokendiagram ziet er als volgt uit:



Afbeelding 1: Het strokendiagram van de speel-o-theek database

En de tabellen zijn:

SPEELGOED
speelgoednummer
speelgoednaam
categorienummer
leenduur_dagen

CATEGORIE
categorienummer
categorienaam

UITLEEN
uitleennummer
speelgoednummer
lidnummer
datum_uitgeleend
datum_terug

LID
lidnummer
voornaam
tussenvoegsel
achternaam
adres
postcode
woonplaats
telefoonnummer
email
lidsoortnummer

LIDSOORT
lidsoortnummer
lidsoortnaam

2.3 Hoe kan het allemaal wat handiger?

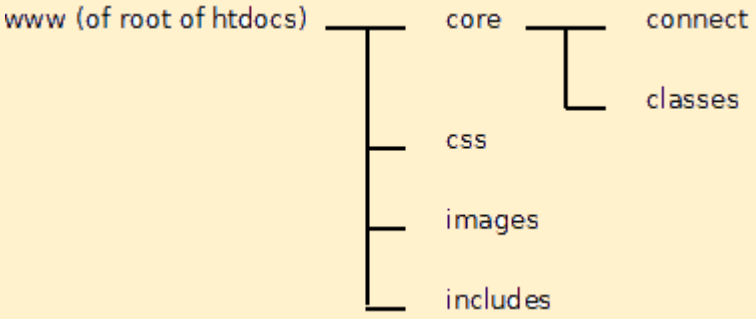
Je hebt inmiddels wel gemerkt dat als je eenmaal aan het programmeren bent, je al snel heel veel bestanden hebt.
Laten we daar eerst eens wat orde in scheppen.

2.3.1 Mappenstructuur

Om te beginnen is het goed een bepaalde mappenstructuur te bedenken. Tot nu toe hebben we daar weinig tijd aan besteed maar je hebt ook wel gemerkt dat je bij het programmeren binnen de kortste keren een behoorlijke chaos aan bestanden kunt hebben.

Opdracht 7

- Laat in de rootmap (dus jouw htdocs map of jouw root of www map) de map (met inhoud) phpliteadmin staan maar maak verder de root-map leeg.
- Maak daarna in de root-map de volgende submappen:



Afbeelding 2: de mappenstructuur

Waar we iedere map voor gaan gebruiken zal in de loop van dit deel wel duidelijk worden. Uiteraard staat het jou vrij daar andere namen voor te gebruiken of er nog andere mappen aan toe te voegen.

2.3.2 De database aanmaken

In het strokendiagram heb je de tabellen die we gaan gebruiken al kunnen zien.
We gaan deze nu eerst in SQLite maken.

Opdracht 8

- Start phpLiteAdmin door WebServer te starten (als die nog niet gestart is) en dan in het browservenster in te typen:
  `http://localhost/phpliteadmin/phpliteadmin.php`
- Typ dan in het gedeelte **Create New Database** het volgende:

- en klik op Create

Daarmee is de lege database gemaakt.
Met behulp van SQL opdrachten gaan we daarin de tabellen maken:

Opdracht 9

- Selecteer en kopieer de onderstaande coderegels
- Ga daarna naar phpLiteAdmin, klik op SQL en plak deze regels in het venster dat is verschenen
- Klik dan op Go en de gegevens worden geïmporteerd
- Klik op Structure en je ziet dat er nu 5 tabellen in de database staan

```
BEGIN TRANSACTION;

----
-- Table structure for categorie
----
CREATE TABLE 'categorie' (
  'categorienummer' INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  'categorienaam' TEXT
);

----
-- Table structure for lid
----
CREATE TABLE 'lid' (
  'lidnummer' INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  'voornaam' TEXT,
  'tussenvoegsel' TEXT,
  'achternaam' TEXT,
  'adres' TEXT,
  'postcode' TEXT,
  'woonplaats' TEXT,
  'telefoonnummer' TEXT,
  'email' TEXT,
  'lidsoortnummer' INTEGER
);

----
-- Table structure for lidsoort
----
CREATE TABLE 'lidsoort' (
  'lidsoortnummer' INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  'lidsoortnaam' TEXT
);

----
-- Table structure for speelgoed
----
CREATE TABLE 'speelgoed' (
  'speelgoednummer' INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  'speelgoednaam' TEXT,
  'categorienummer' INTEGER,
  'leenduur_dagen' INTEGER
);

----
-- Table structure for uitleen
----
CREATE TABLE 'uitleen' (
  'uitleennummer' INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  'speelgoednummer' INTEGER,
```

```
  'lidnummer' INTEGER,
  'datum_uitgeleend' DATETIME,
  'datum_terug' DATETIME
);

COMMIT;
```

2.3.3 Verbinden met de database

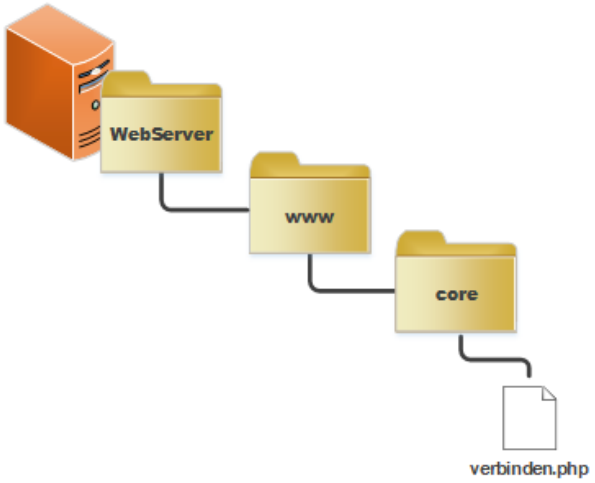
Als we gebruik willen maken van de database moeten we verbinding maken met SQLite. Zoals je al eerder hebt kunnen zien gebruiken we dat erg vaak dus laten we daarvoor een php-programma voor maken.

Opdracht 10

- Maak een programma met de naam **verbinden.php**, plaats dit bestand in de map **core/connect** en zet daar de volgende code in:

```
<?php
  // Maken van verbinding
  $db = new PDO("sqlite:../data/speelotheek.sqlite");
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>
```

Geen nieuws in dit programma want dat heb je allemaal in het eerste deel van deze reader gehad. Wel belangrijk is dat dit programma in de juiste map komt te staan!

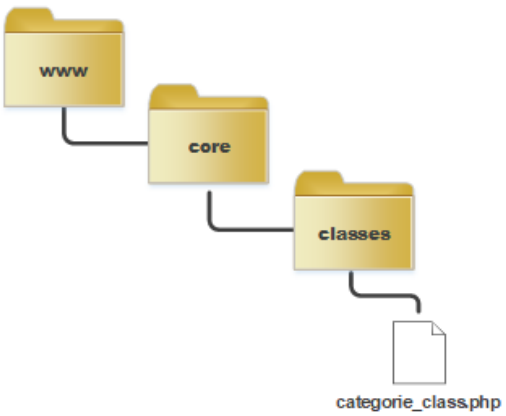


Afbeelding 3: Verbinden

We hoeven dus, iedere keer als we gebruik willen maken van de database, alleen maar dit stukje programma op te roepen.
Maar hoe doen we dat dan? Dat komt. Laten we eerst eens kijken hoe we een eenvoudige tabel als **categorie** kunnen gaan benaderen.

2.3.4 Een klasse

Een klasse van een tabel is (in ons geval) een beschrijving van alle functies die we nodig zullen hebben om de informatie uit die tabel te halen, nieuwe informatie toe te voegen of de informatie te wijzigen. Klinkt nog niet simpel hè?
Geduld!



Afbeelding 4: Classes

In opdracht 11 gaan we eerst de klasse voor de tabel categorie maken.
In deze klasse leggen we later de volgende methoden vast:

- Het toevoegen van een nieuwe categorie
- Het opvragen van een categorie als we het categorienummer weten
- Het wijzigen van een categorie
- Het verwijderen van een categorie
- Een overzicht van alle categorieën
- Het aantal categorieën
- Het controleren of een categorie al bestaat
- Het controleren of er een categorie is met dezelfde naam maar een ander categorienummer

Maar we beginnen eenvoudig.

Opdracht II

- Neem onderstaande code over, geef die de naam **categorie_class.php** en sla deze op in de map **core/classes**

```php
<?php
  class Categorie {

    private $db;

    /*-------------------------------------------------------
     * De constructor
     * Als we de klasse Categorie willen gaan gebruiken
     * maken we (in init.php) een object variabele aan
     * met behulp van de constructor.
     * Daarna kunnen we gebruik maken van de functies
     * in deze klasse
     -------------------------------------------------------*/
    public function __construct($database) {
      $this->db = $database;
    }

    /*-------------------------------------------------------
     * met deze functie krijgen we een overzicht
     * van het aantal aanwezige categorieën
     -------------------------------------------------------*/
    public function aantal_categorie() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM categorie"
      );

      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }
?>
```

In de klasse Categorie hebben we nu nog maar een functie opgenomen, namelijk om het aantal categorieën in de database te bepalen. Later komen de andere functies.

2.3.5 De initialisatie

Nu wordt het tijd het programma op te bouwen waarmee we deze klasse kunnen gebruiken.
In de code van de klasse Categorie (bij de constructor) wordt al gerept over het programma **`init.php`**. Laten we dat programma eerst maar even gaan maken.

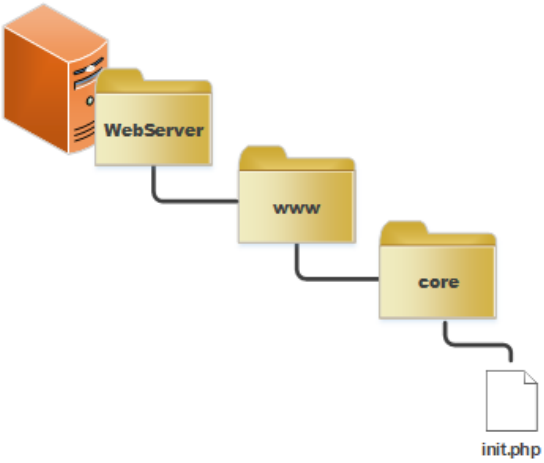
Opdracht 12

Nu gaan we zorgen, dat we via het bestand **`init.php`** verbinding maken met de klasse **`Categorie`**.

- Neem onderstaande code over, geef die de naam **`init.php`** en sla deze op in de map **`core`**.

```php
<?php
  require 'connect/verbinden.php';
  require 'classes/categorie_class.php';
  $categorie = new Categorie($db);
  $errors = array();
?>
```

Kijk maar eens goed naar de code van **`init.php`**. Want hier gebeurt het eigenlijk. Hier wordt de verbinding gelegd met de database en wordt de klasse **`categorie.php`** ingelezen.
Daarna wordt met de regel **`$categorie = new Categorie($db);`** een nieuw object van de klasse Categorie aangemaakt. Daarmee kunnen we later in de code dat object steeds gebruiken door **`$categorie`** aan te roepen.

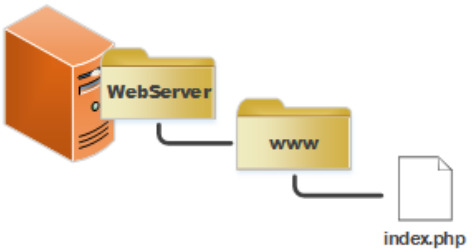


Afbeelding 5: Initialisatie

Alles wat we tot nu toe gemaakt hebben staat in de map core of de submappen daarvan. Kortom, de kern van ons database programma is hiermee gelegd. Uiteraard is het nog lang niet compleet maar van hieruit kunnen we verder bouwen.

2.3.6 Het programma

Zoals gewoonlijk begint een php-programma met het bestand **index.php**



Afbeelding 6: Index

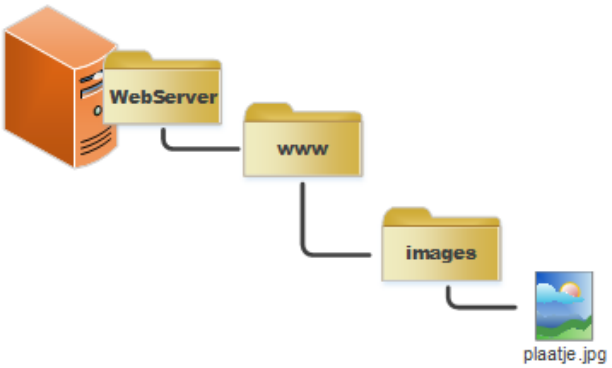
Opdracht 13

- Neem onderstaande code over, geef die de naam **index.php** en sla deze op in de root-map.

```
<?php
require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">
```

```
  <head>
    <meta charset="utf-8">
    <link href='http://fonts.googleapis.com/css?family=Princess+Sofia'
rel='stylesheet' type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Speel-o-theek</title>
  </head>

  <body>
    <div id="container">
      <img class="center" src="images\speelotheek.png" />
      <?php
        echo '<div>';
        include 'includes/menu.php';
        echo '</div>';
        echo '<ul>';
        echo '<li>';
        $aantalcategorie = $categorie->aantal_categorie();
        if ($aantalcategorie > 0) {
          echo "Aantal categorieën ".$aantalcategorie;
        }
        else echo "Er zijn geen categorieën in de database.<br>";
        echo '</li>';
        echo '</ul>';
      ?>
    </div>
  </body>
</html>
```

In deze code wordt er gebruik gemaakt van het plaatje `speelotheek.png`. Je kunt die afbeelding **hier** downloaden. Maar uiteraard mag je ook een andere toepasselijke afbeelding kiezen. Plaats de afbeelding in de map **`images`**.



Afbeelding 7: Images

Eigenlijk de belangrijkste regel in deze code is de tweede. Daarin wordt **`init.php`** opgeroepen en op zijn beurt zorgt **`init.php`** voor de verbinding met de database en voor het maken van objecten van de klassen.

Ook bijzonder in deze code is een verwijzing naar een Google font. Google heeft heel veel lettertypes die je in jouw website mag gebruiken.
Kijk maar eens op: **google.com/fonts**
Je ziet op die website ook hoe je zo'n font kunt gebruiken.

Als je goed naar de code gekeken hebt, dan heb je gezien dat er nog twee bestanden ontbreken, namelijk het stylesheet **`style.css`** en het menu bestand, **`menu.php`**. Dus die gaan we in de volgende twee opdrachten nog even maken.

Opdracht 14

- Neem onderstaande code over, geef die de naam **`style.css`** en sla deze op in de map **`css`**.

```
@charset "utf-8";
/* CSS Document */

body {
  font: 0.9em Tahoma, Verdana, Arial;
  line-height:172%;
  background: #ddd;
}

.center {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

#container {
  width: 1000px;
  padding: 25px;
  background: #fff;
  margin: 100px auto 0 auto;
  min-height: 500px;
}

.afbeelding_container {
  position: relative;
  float: left;
  margin-left: 25px;
}
```

```css
.afbeelding_container .tekst_container {
  position: absolute;
  top: 25px;
  left: 50px;
  color: #FFFFFF;
  font-family: 'Princess Sofia', cursive;
  font-size: 36px;

}

.schoon {
  clear: both;
}


#overzicht {
  margin-left: 25px;
}

.links {
  float: left;
  width: 320px;
}

#rechts {
  float: right;
}

#onder {
  clear: both;
}

.breder {
  width: 250px;
}

ul {
  padding-left: 35px;
  padding-right: 35px;
  list-style: none;
}

hr {
  margin: 12px 0;
  height: 1px;
  border: 1px solid #fff;
  border-top: 1px solid #ccc;
  background-color: #fff;
}
```

```css
a:link {
  color: blue;
  text-decoration: none;
}

a:visited {
  color: blue;
  text-decoration: none;
}

a:hover {
  color: black;
  text-decoration: none;
}

a:active {
  color: blue;
  text-decoration: none;
}

.menu {
    margin:auto;
    padding:30px;
}

ul.menu {
    list-style-type: none
}

.menu li {
    float:left;
    position:relative;
    width: 185px;
    text-align:center;
    text-decoration: none;
    margin: 0;
    padding: 0;
}

.menu li a {
    display: block;
    padding-bottom: 20px;
    padding-right: 10px;
    padding-top: 10px;
    padding-left: 10px;
    text-decoration: none;
    position: relative;
    z-index: 100;
    -webkit-transition: all 1s;
```

```
    -moz-transition: all 1s;
    -o-transition: all 1s;
    transition: all 1s;
}

.menu li a span{
  display: block;
  padding-top: 10px;
  font-weight: 700;
  font-size: 20px;
  color: rgba(120, 120, 120, 0.9);
  font-family: 'Princess Sofia', cursive;
  font-size: 28px;
}

.menu li:hover span{
  color: #fff;
}

.menu li:nth-child(1):hover a{
  background-color: rgba(175,54,55,0.8);
  -moz-transform: rotate(3deg);
  -webkit-transform: rotate(3deg);
  -o-transform: rotate(3deg);
  transform: rotate(3deg);
}

.menu li:nth-child(2):hover a{
  background-color: rgba(199, 204, 73, 0.8);
  -moz-transform: rotate(-3deg);
  -webkit-transform: rotate(-3deg);
  -o-transform: rotate(-3deg);
  transform: rotate(-3deg);
}
.menu li:nth-child(3):hover a{
  background-color: rgba(213, 135, 11, 0.8);
  -moz-transform: rotate(3deg);
  -webkit-transform: rotate(3deg);
  -o-transform: rotate(3deg);
  transform: rotate(3deg);
}
.menu li:nth-child(4):hover a{
  background-color: rgba(51, 143, 144, 0.8);
  -moz-transform: rotate(-3deg);
  -webkit-transform: rotate(-3deg);
  -o-transform: rotate(-3deg);
  transform: rotate(-3deg);
}
.menu li:nth-child(5):hover a{
  background-color: rgba(117,18,98,0.8);
```

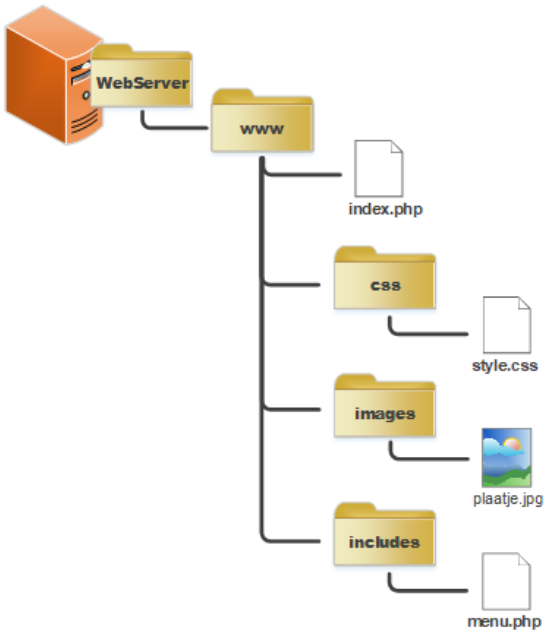
```
  -moz-transform: rotate(3deg);
  -webkit-transform: rotate(3deg);
  -o-transform: rotate(3deg);
  transform: rotate(3deg);
}
.menu li:nth-child(6):hover a{
  background-color: rgba(33, 136, 215, 0.8);
  -moz-transform: rotate(-3deg);
  -webkit-transform: rotate(-3deg);
  -o-transform: rotate(-3deg);
  transform: rotate(-3deg);
}
.menu li:nth-child(7):hover a{
  background-color: rgba(109, 109, 109, 0.8);
  -moz-transform: rotate(3deg);
  -webkit-transform: rotate(3deg);
  -o-transform: rotate(3deg);
  transform: rotate(3deg);
}
.menu li:nth-child(8):hover a{
  background-color: rgba(152, 120, 92, 0.8);
  -moz-transform: rotate(-3deg);
  -webkit-transform: rotate(-3deg);
  -o-transform: rotate(-3deg);
  transform: rotate(-3deg);
}

th {
  padding: 10px 30px 10px 30px;
}

td {
  padding: 0 30px 0 30px;
}

td.muteren {
  padding: 0 0 0 10px;
}

tbody:before {
  line-height:1em;
  display:block;
}

thead {
  text-align: left;
}
```

Niet alle code in dit css bestand wordt al gebruikt maar dat komt in het vervolg

Opdracht 15

- Neem onderstaande code over, geef die de naam **menu.php** en sla deze op in de map **includes**.

```
<ul class="menu">
  <li><a href="speelgoed.php"><span>Speelgoed</span></a></li>
  <li><a href="uitlening.php"><span>Uitleningen</span></a></li>
  <li><a href="lid.php"><span>Leden</span></a></li>
  <li><a href="categorie.php"><span>Categorie</span></a></li>
  <li><a href="lidsoort.php"><span>Lidsoort</span></a></li>
</ul>
<br><br>
<hr>
```



Afbeelding 8: Website

Zo, klaar voor de eerste test:

Opdracht 16

- Start het programma (vooropgesteld dat webserver draait) met de opdracht **localhost/index.php** (of gewoon **localhost**).

- Als het goed gegaan is zie je nu het startscherm zoals in afbeelding 9. Gelukt?

Afbeelding 9: Het startscherm

We hebben nu al heel veel programma's. Tijd om het weer even op een rijtje te zetten.

- We hebben een database met 5 tabellen. Daarvan hebben we ons eerst gericht op de tabel **`categorie`**.
- Om verbinding te maken met die database gebruiken we **`core/connect/verbinden.php`**
- Alle manieren om de tabel **`categorie`** te kunnen benaderen, aan te passen en opvragingen te doen beschrijven we in **`core/classes/categorie_class.php`** maar nu hebben we daar alleen nog maar de methode **`aantal_categorie()`** in gemaakt.
- Met het programma **`core/init.php`** zorgen we voor het initialiseren zoals het maken van de verbinding met de database en het gebruiken de objectvariabele **`$categorie`** als het "aanspreekpunt" om de methoden van de klasse **`Categorie`** te gebruiken.
- En dit alles komt samen in het programma **`index.php`**. Daar wordt er nog een css-bestand aan gekoppeld en ook nog even het menu ingelezen.

Het wordt natuurlijk pas echt leuk wanneer we ook wat gegevens in de database hebben zitten. Maar we zijn nog wel even bezig voor we de complete crud applicatie voor de categorie tabel gemaakt hebben dus als je toch wilt weten of het wel echt werkt, voer dan een categorie via phpLiteAdmin in. (Let op: vul dan geen categorienummer in, dat doet hij zelf. **`AUTOINCREMENT`**, weet je nog wel?).

Opdracht 17

- Maak nu, met de klasse **categorie_class.php** als voorbeeld ook de klassen **speelgoed_class.php**, **lid_class.php**, **lidsoort_class.php** en **uitleen_class.php** aan.
- Maak in ieder van die klassen de methode aan om het aantal op te vragen
- Pas **init.php** aan zodat de klassen met **require** worden aangeroepen
- Pas **init.php** aan zodat de objectvariabelen **$speelgoed**, **$lid**, **$lidsoort** en **$uitleen** worden aangemaakt
- Pas **index.php** dusdanig aan dat ook van de tabellen speelgoed, lid, lidsoort en uitleen het aantal records wordt getoond.

Nog even een voorbeeld van zo'n klasse?
Ok, hier de klasse speelgoed. Je ziet wel dat er maar heel weinig verschil is met de klasse categorie maar als de functies wat uitgebreider worden zit er wel meer verschil tussen.

```php
<?php
  class Speelgoed {

    private $db;

    /*-------------------------------------------------
     * De constructor
     -------------------------------------------------*/
    public function __construct($database) {
      $this->db = $database;
    }

    /*-------------------------------------------------
     * met deze functie krijgen we een overzicht van het
     * aantal stuks aanwezige speelgoed
     -------------------------------------------------*/
    public function aantal_speelgoed() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM speelgoed"
```

```
        );

        try{
          $query->execute();
          $rows = $query->fetchColumn();
          return $rows;
        }catch(PDOException $e){
          die($e->getMessage());
        }
      }
    }
?>
```

Als je opdracht 17 klaar hebt ziet het er ongeveer uit als in afbeelding 10.

Afbeelding 10: Opdracht 17 uitgewerkt

2.3.7 Het programma deel 2

Even een afspraak om het overzichtelijk te houden. Dit is belangrijk omdat je een aantal bestanden krijgt met min of meer dezelfde naam.
Voor een klassebestand gebruiken we bij de naamgeving enkelvoud en daar aan toegevoegd **`_class`**, dus bijvoorbeeld **`categorie_class.php`**.
Voor de programmabestanden gebruiken we de volgende naamgeving:

`categorie.php`
`categorie_muteren.php`

categorie_toevoegen.php
categorie_verwijderen.php
categorie_wijzigen.php

Dit doen we op dezelfde manier met alle andere programmabestanden.
Wat die afzonderlijke programma's doen volgt zo wel.
Maar je zult zien dat als je dat consequent aan houdt, het met die veelheid aan bestanden toch nog overzichtelijk blijft.

Als je, in jouw uitgewerkte voorbeeld van opdracht 17 op Categorie klikt, krijg je nog een foutmelding. Logisch want in **menu.php** verwijst deze link naar het bestand **categorie.php** en die bestaat nog niet.
Dus daar gaan we nu mee aan de slag.

Opdracht 18

- Neem onderstaande code over, geef die de naam **categorie.php** en sla deze op in de root-map.

```
<?php
  require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">

  <head>
    <meta charset="utf-8">
    <link href='http://fonts.googleapis.com/css?family=Princess+Sofia'
rel='stylesheet' type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Categorieën</title>
  </head>
  <body>
    <div id="container">

      <?php
        include 'includes/categorie_kop.php';
        include 'includes/categorie_menu.php';

        echo '<ul>';
        echo '<li>';

        $aantalcategorie = $categorie->aantal_categorie();
        if ($aantalcategorie > 0) {
          echo $aantalcategorie." categorieën";
```

```
        } else {
          echo "Er zijn geen categorieën. <br>In het menu hierboven kan je
nieuwe categorieën toevoegen.";
        }

        echo '</li>';
        echo '</ul>';
      ?>

    </div>
  </body>
</html>
```

Je ziet dat het een vrijwel exacte kopie van **index.php** is. Alleen moet er een ander menu komen.
Verder wordt er een bestand **categorie_kop.php** toegevoegd.
Die bestanden gaan we nu maken.

Opdracht 19

- Neem onderstaande code over, geef die de naam **categorie_menu.php** en sla deze op in de map **includes**.

```
<ul class="menu">
  <li><a href="index.php"><span>Home</span></li></a>
  <li><a href="categorie_toevoegen.php"><span>Toevoegen</span></a></li>
  <li><a href="categorie_muteren.php"><span>Overzicht</span></a></li>
</ul>
<br><br>
<hr>
```

Opdracht 20

- Neem onderstaande code over, geef die de naam **categorie_kop.php** en sla deze op in de map **includes**.

```
<div class='afbeelding_container'>
  <img class='center' src='images\speelgoed-vrachtauto4.png'>
  <div class='tekst_container'>
    Categorie
  </div>
</div>

<div class='schoon'>
```

```
</div>
```

In **categorie_kop.php** heb je weer een afbeelding nodig. Die kun je **hier** downloaden.

Als je nu in het hoofdmenu op Categorie klikt zal afbeelding 11 het resultaat (moeten) zijn:

Afbeelding 11: Opdracht 19 en 20 uitgewerkt

Net als het programma **categorie_menu.php** is **categorie_kop.php** een programma dat we meerdere keren zullen gaan gebruiken. Vandaar dat we een apart programma gebruiken dat we dan simpel kunnen **include**'n. All zou de deze beide programma's ook kunnen samenvoegen.

Voor de categorieën hebben we nu 3 menukeuzes.
De keuze Home is al ingevuld. Die gaat gewoon weer terug naar **index.php**.
Maar met de andere 2 keuzes gaan we nu aan de slag.

2.3.7.1 Categorie toevoegen

Wat komt er allemaal bij kijken als we een categorie toe willen voegen?

- Allereerst zullen we een formulier in HTML moeten maken waar we de naam van het genre in kunnen vullen. Daarna volgen enkele controles.
- Zo mag het veld natuurlijk niet leeg zijn, aan een leeg genre hebben we niets.
- Maar ook moeten we gaan controleren of het genre al bestaat want ook dat is niet de bedoeling.
- Tenslotte moeten we in de klasse een functie maken waarmee we het genre kunnen toevoegen en we moeten er een melding van krijgen dat het gelukt is.

Kortom, best een ingewikkelde code.

Opdracht 21

- Om te beginnen hebben we twee nieuwe functies nodig in de klasse **categorie_class.php**
- Voeg de onderstaande functies toe aan **categorie_class.php**. Het maakt niet uit waar je ze plaatst, zolang ze maar binnen de accolades van **class Categorie** komen.

```php
/*-------------------------------------------------------
 * met deze functie kunnen we een nieuwe
 * categorie toevoegen
 -------------------------------------------------------*/
public function toevoegen_categorie($categorienaam){
  $query = $this->db->prepare(
    "INSERT INTO categorie
    (categorienaam) VALUES (?) ");

  $query->bindValue(1, $categorienaam);

  try{
    $query->execute();
  }catch(PDOException $e){
    die($e->getMessage());
  }
}


/*-------------------------------------------------------
 * met deze functie kunnen we controleren
 * of een categorie al in de tabel voorkomt
 -------------------------------------------------------*/
public function bestaatAl_categorie($categorienaam) {
  $query = $this->db->prepare(
    "SELECT COUNT(*)
    FROM categorie
    WHERE categorienaam = ?"
  );

  $query->bindValue(1, $categorienaam);

  try{
    $query->execute();
    $rows = $query->fetchColumn();
    if ($rows > 0){
      return true;
    } else {
      return false;
```

```
    }
  }catch(PDOException $e){
    die($e->getMessage());
  }
}
```

Opdracht 22

- Neem onderstaande code over, geef die de naam `categorie_toevoegen.php` en sla deze op in de root-map.

```
<?php
require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">

  <head>
    <meta charset="utf-8">
    <link href='http://fonts.googleapis.com/css?family=Princess+Sofia'
rel='stylesheet' type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Categorie toevoegen</title>
  </head>

  <body>
    <div id="container">

      <?php
        include 'includes/categorie_kop.php';
      include 'includes/categorie_menu.php';

      /*-------------------------------------
      De code bestaat uit twee gedeelten.
      Als het programma voor het eerst gestart wordt
      is $_POST nog leeg.
      Dat betekent dat hij eerst een groot deel
      van de code overslaat en begint bij het
      formulier (zie verderop).

      Als $_POST niet leeg is betekent dat,
      dat er op de knop Toevoegen van het
      formulier geklikt is.
      -------------------------------------*/
```

```php
    if (!empty($_POST)) {
      $bestaatal = $categorie->bestaatAl_categorie($_POST['categorienaam']);
      if ($bestaatal) {

        // Als de categorie al bestaat mag deze niet toegevoegd worden
        $errors[] = 'Deze categorie bestaat al';
      } else if (empty($_POST['categorienaam'])){

        // Als het veld $_POST['categorienaam'] leeg
        // is moet er een foutmelding volgen.
        $errors[] = 'De categorienaam moeten worden ingevuld';

      } else {

        // Als aan de voorwaarden niet leeg en bestaat nog niet voldaan
        // is kan de naam worden toegevoegd
        $categorie->toevoegen_categorie($_POST['categorienaam']);

        // Nu de categorie is toegevoegd, voorzien we dit programma
        // van een $_GET variabele door die achter de programmanaam
        // te schrijven met een ? ertussen
        header('Location: categorie_toevoegen.php?success');

      }
    }

    // Bestaat de $_GET['success'] variabele?
    // Dan kunnen we melden dat de categorie is toegevoegd
    if (isset($_GET['success']) === true && empty($_GET['success']) === true)
{
      echo '<p>De categorie is toegevoegd</p>';
    } else if (!empty($errors)) {

      // En anders wordt gekeken naar de variabele $errors. Daarin worden
      // eventuele foutmeldingen verzameld.
      // Die foutmeldingen worden hier afgedrukt.
      echo '<p>' . implode('</p><p>', $errors) . '</p>';

    } else {
      ?>

      <!--------------------------------------
      Als $_POST bij het starten van het programma leeg is zal hij
      meteen naar het formulier hieronder gaan en kun je een nieuwe
      categorie invoeren.
      --------------------------------------->

      <h2>Voeg een categorie toe</h2>
      <hr>
```

```
        <form action="" method="post">
          <ul>
            <li>
              <h4>Categorienaam:</h4>
              <input type="text" name="categorienaam" size="30"
placeholder="categorienaam">
            </li>
            <li>
              <br>
              <input type="submit" value="Toevoegen">
            </li>
          </ul>
        </form>

        <!--------------------------------------
        Einde van het formulier.
        Als er op de knop toevoegen geklikt is
        start dit programma opnieuw met de
        ingevulde genrenaam opgeslagen in $_POST
        -------------------------------------->

        <?php
      }
      ?>

    </div>
  </body>
</html>
```

Best wel een ingewikkeld programma omdat er een aantal verschillende dingen gebeuren en diverse controles plaatsvinden.
Probeer aan de hand van de commentaarregels goed te begrijpen wat de afzonderlijke delen van de code doen.
Als de code goed werkt kun je hiermee nieuwe categorieën toevoegen. Hoewel je ze nog niet kunt zien, merk je het resultaat wel aan de hand van het aantal categorieën wat steeds bij het menu staat.
Maar laten we nu ook meteen maar een overzicht van de aanwezige categorieën maken. Gelukkig is dat minder lastig.

2.3.7.2 Categorieën bekijken, wijzigen en verwijderen

Voor het bekijken van de categorieën moeten we eerst in de klasse weer een functie toevoegen.
Daarna kunnen we het programma maken om het overzicht op het scherm te zetten.

Opdracht 23

- Om te beginnen hebben we weer een nieuwe functie nodig in de klasse **categorie_class.php**.
- Voeg de onderstaande functie toe aan **categorie_class.php**. Het maakt niet uit waar je hem plaatst, zolang hij maar binnen de accolades van **class Categorie** komen.

```
  /*-----------------------------------------------------
   * met deze functie krijgen we een overzicht
   * van alle aanwezige categorieën
   -----------------------------------------------------*/
  public function overzicht_categorie() {
    $query = $this->db->prepare(
      "SELECT *
      FROM categorie
      ORDER BY categorienaam"
    );

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
    return $query->fetchAll();
  }
```

Herken je de SQL opdracht om alle gegevens uit de tabel categorie te halen?

Opdracht 24

- Neem onderstaande code over, geef die de naam **categorie_muteren.php** en sla deze op in de root-map.

```
<?php
  require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <link href='http://fonts.googleapis.com/css?family=Princess+Sofia'
rel='stylesheet' type='text/css'>
```

```
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Categorieën overzicht</title>
  </head>

  <body>
    <div id="container">

      <?php
        include 'includes/categorie_kop.php';
        include 'includes/categorie_menu.php';
      ?>

      <div id="overzicht">
        <table>
          <thead>
            <tr>
              <th>Categorienummer</th>
              <th>Categorienaam</th>
              <th></th>
              <th></th>
            </tr>
          </thead>

          <tbody>
            <?php
              $resultaat = $categorie->overzicht_categorie();
              foreach ($resultaat as $rij) {
                echo '<tr>';
                echo '
                  <td>'.$rij['categorienummer'].'</td>
                  <td>'.$rij['categorienaam'].'</td>
                ';
                echo '
                  <td class="muteren">
                  <form action="categorie_wijzigen.php" method="post">
                    <input type="image" src="images/modify.png">
                    <input type="hidden" name="nummer"
value="'.$rij['categorienummer'].'">
                  </form>
                </td>
              ';
              echo '
                <td class="muteren">
                  <form action="categorie_verwijderen.php" method="post">
                    <input type="image" src="images/delete.png">
                    <input type="hidden" name="nummer"
value="'.$rij['categorienummer'].'">
                  </form>
                </td>
              ';
```

```
            echo '</tr>';
            }
          ?>
        </tbody>
      </table>
    </div>
  </div>
 </body>
</html>
```

Je hebt nog wel twee plaatjes nodig.
Je kunt modify.png **hier** en delete.php **hier** downloaden

Het resultaat zie je in afbeelding 6.

Afbeelding 6: Het overzicht van de categorieën

Opdracht 25

Vind je de nummers voor het overzicht niet mooi?

- Zoek uit hoe je die weg kunt laten.

Uiteraard zul je wel in de gaten hebben dat staat voor wijzigen en voor verwijderen.
Beiden staan ze in een `<form>` met een `action` die bij wijzigen verwijst naar `categorie_wijzigen.php` en bij verwijderen naar `categorie_verwijderen.php`.

Bij het doorlopen van de records van de categorie-tabel maakt hij voor ieder record twee keer een `<form>` aan die beide in een cel van de tabel komen. In dat `<form>` staat dan het plaatje en met `type="hidden"` het nummer van de categorie.
Op die manier wordt het categorienummer van de categorie die we willen wijzigen via `$_POST` doorgegeven aan `categorie_wijzigen.php` of `categorie_verwijderen.php`
Bij het aanklikken van het icon op een regel wordt de `action` van dat betreffende `<form>` uitgevoerd en wordt het nummer via de methode `post` doorgegeven.

De programma's `categorie_wijzigen.php` en `categorie_verwijderen.php` moeten we dus nog maken om alle CRUD acties te kunnen uitvoeren.

Maar we beginnen met de methoden die we nog aan de klasse toe moeten voegen om dit te kunnen laten werken.

Opdracht 26

Voor het wijzigen en verwijderen hebben we 4 nieuwe functies nodig in de klasse **`categorie_class.php`**.

- Voeg de onderstaande functies toe aan **`categorie_class.php`**. Het maakt niet uit waar je ze plaatst, zolang ze maar binnen de accolades van `class` **`Categorie`** komen.

```php
/*-------------------------------------------------
 * met deze functie kunnen we een categorie
 * opvragen met behulp van het categorienummer
 -------------------------------------------------*/
public function opvragen_categorie($categorienummer) {
  $query = $this->db->prepare(
    "SELECT *
    FROM categorie
    WHERE categorienummer = ?"
  );

  $query->bindValue(1, $categorienummer);

  try{
    $query->execute();
  }catch(PDOException $e){
    die($e->getMessage());
  }
  return $query->fetchAll();
}
```

```
  /*-----------------------------------------------------
   * met deze functie kunnen we een categorie wijzigen
   -----------------------------------------------------*/
  public function wijzigen_categorie($categorienaam, $categorienummer) {
    $query = $this->db->prepare(
      "UPDATE categorie
      SET categorienaam = ?
      WHERE categorienummer = ?"
    );

    $query->bindValue(1, $categorienaam);
    $query->bindValue(2, $categorienummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }

  /*-----------------------------------------------------
   * met deze functie kunnen we een categorie verwijderen
   -----------------------------------------------------*/
  public function verwijderen_categorie($categorienummer) {
    $query = $this->db->prepare(
      "DELETE FROM categorie
      WHERE categorienummer = ?"
     );

    $query->bindValue(1, $categorienummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }

  /*-----------------------------------------------------
   * met deze functie kunnen we tijdens het wijzigen
   * controleren of een categorie al in de tabel
   * voorkomt maar hierbij wordt gecontroleerd of de
   * mogelijk al bestaande categorie wel een ander
   * categorienummer heeft
   -----------------------------------------------------*/
  public function bestaatAlMetAnderNummer_categorie($categorienaam,
$categorienummer) {
    $query = $this->db->prepare(
      "SELECT COUNT(*)
      FROM categorie
```

```
      WHERE categorienaam = ?
                    AND categorienummer <> ?"
    );

    $query->bindValue(1, $categorienaam);
    $query->bindValue(2, $categorienummer);

    try{
      $query->execute();
      $rows = $query->fetchColumn();
      if ($rows > 0){
        return true;
      } else {
        return false;
      }
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }
```

Dan nu de code om een categorie te wijzigen:

Opdracht 27

- Neem onderstaande code over, geef die de naam **categorie_wijzigen.php** en sla deze op in de root-map.

```
<?php
  require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">

  <head>
    <meta charset="utf-8">
    <link href='http://fonts.googleapis.com/css?family=Princess+Sofia'
rel='stylesheet' type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Categorie wijzigen</title>
  </head>

  <body>
    <div id="container">
      <?php
        include 'includes/categorie_kop.php';
        include 'includes/categorie_menu.php';
```

```
        /*-------------------------------------
        Als we dit programma starten vanuit
        categorie_muteren.php krijgt $_POST een waarde
        mee. Maar die waarde zit in $_POST['nummer'].

        Maar als we dit programma starten vanuit het
        formulier verderop in dit programma bestaat
        $_POST['categorienummer'].

        Met dit gegeven kunnen we besluiten wat
        het programma moet doen.
        -------------------------------------*/

        if (!empty($_POST) && isset($_POST['categorienummer'])) {

          /* ------------------------------
          Het programma wordt dus gestart vanuit het formulier hieronder
          want $_POST['categorienummer'] bestaat (isset).
          Dus het formulier is ingevuld. Nu kan de controle volgen.
          Als het veld categorienaam leeg is moet er een foutmelding volgen.
          -------------------------------------*/

          if (empty($_POST['categorienaam'])){
            $errors[] = 'Categorienaam mag niet leeg zijn';
          } else {

            /*-------------------------------------
            De controle of de categorienaam al bestaat
            is nu een stukje ingewikkelder.
            Deze categorienaam bestaat immers al want we
            zijn juist deze categorie aan het wijzigen.

            Als we alleen op categorienaam controleren
            moeten we dus controleren of die gegevens al
            in de database zitten bij een ander categorienummer.

            Vandaar dat we een andere bestaatAl_categorie
            methode moeten hebben.
            We gaan zo nog een nieuwe methode in de
            klasse aanmaken die we
            bestaatAlMetAnderNummer_categorie noemen.

            De rest van de code kun je herkennen van
            categorie_toevoegen.php
            -------------------------------------*/

            $bestaatal =
$categorie->bestaatAlMetAnderNummer_categorie($_POST['categorienaam'],
$_POST['categorienummer']);
```

```
            if ($bestaatal) {
              $errors[] = 'Deze gegevens staan al bij een andere categorie in de
database';
            } else {
              $categorie->wijzigen_categorie($_POST['categorienaam'],
$_POST['categorienummer']);
              header('Location: categorie_muteren.php?success&mut=1');
            }
          }

          if (isset($_GET['success']) === true && empty($_GET['success']) ===
true) {
            echo '<p>De categorie is gewijzigd</p>';
          } else if (!empty($errors)) {
            echo '<p>' . implode('</p><p>', $errors) . '</p>';
          }
        } else {
          $categorienummer = $_POST['nummer'];
          $resultaat = $categorie->opvragen_categorie($categorienummer);
          foreach ($resultaat as $rij) {
            $categorienaam = $rij['categorienaam'];
          }
        ?>
        <h2>Wijzig de categorie</h2>
        <hr>

        <form action="" method="post">
          <ul>
            <li><h4>Speelgoedcategorie:</h4></li>
            <li><input type="text" name="categorienaam" size="40"
value='<?=$categorienaam?>'></li>
              <input type='hidden' name='categorienummer'
value='<?=$categorienummer?>'>
              <li><input type="submit" value="Wijzigen"></li>
            </ul>
          </form>

          <?php
        }
      ?>

    </div>
  </body>
</html>
```

Op zich staat er in opdracht 27 niet zoveel vreemds. Het meeste heb je eerder gehad en veel code is gelijk aan die **categorie_toevoegen.php**.

In het form zie je ook een andere manier om php-variabelen in HTML te gebruiken, namelijk zo: `<?=$categorienaam?>` en `<?=$categorienummer?>`.
Lekker beknopt en het houdt de code overzichtelijk.

We hebben in opdracht 26 ook al de methode voor het verwijderen van een categorie toegevoegd dus nu alleen nog het programma **categorie_verwijderen.php**

Opdracht 28

- Neem onderstaande code over, geef die de naam **categorie_verwijderen.php** en sla deze op in de root-map.

```
<?php
  require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <link href='http://fonts.googleapis.com/css?family=Princess+Sofia'
rel='stylesheet' type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Categorie verwijderen</title>
  </head>
  <body>
    <div id="container">
      <?php
      include 'includes/categorie_kop.php';
      include 'includes/categorie_menu.php';

      /*-------------------------------------
      Als we dit programma starten vanuit
      categorie_muteren.php krijgt $_POST een waarde
      mee. Maar die waarde zit in $_POST['nummer'].

      Maar als we dit programma starten vanuit het
      formulier verderop in dit programma bestaat
      $_POST['categorienummer'].

      Met dit gegeven kunnen we besluiten wat
      het programma moet doen.
      -------------------------------------*/

      if (!empty($_POST) && isset($_POST['categorienummer'])) {
```

```
        /* ------------------------------
        Het programma wordt dus gestart vanuit het
        formulier hieronder want $_POST['categorienummer']
        bestaat (isset).

        Is er inderdaad op de knop verwijderen geklikt?
        Dat kunnen we controleren met
        isset($_POST['verwijderen'])
        -------------------------------------*/

        if (isset($_POST['verwijderen'])) {

          // Dan kunnen we hem verwijderen.

$categorie->verwijderen_categorie($_POST['categorienummer']);
          echo '<p>De categorie '.$_POST['categorienaam'].' is verwijderd</p>';
        } else {
          // Niets verwijderd?
          // Dan terug naar het categorie overzicht.
          header('Location: categorie_muteren.php?mut=2');
        }

      } else {
        $categorienummer = $_POST['nummer'];
        $resultaat = $categorie->opvragen_categorie($categorienummer);
        foreach ($resultaat as $rij) {
          $categorienaam = $rij['categorienaam'];
        }
        ?>
        <h2>Verwijder deze categorie</h2>
        <hr>

        <form action="" method="post">
          <ul>
            <li><h4>Categorie:</h4></li>
            <li><?=$categorienaam?></li>
              <input type='hidden' name='categorienummer'
value='<?=$categorienummer?>'>
              <input type='hidden' name='categorienaam'
value='<?=$categorienaam?>'>
            <li><br>Weet je zeker dat je deze categorie wilt verwijderen?</li>
            <li>
              <input type='submit' name='verwijderen' value='Verwijderen'>
              <input type='submit' value='Terug'>
            </li>
          </ul>
        </form>

        <?php
```

```
      }
      ?>

    </div>
  </body>
</html>
```

En daarmee is de crud applicatie voor wat betreft de categorieën klaar.
Maar uiteraard zijn we er nog niet. We hebben immers 5 tabellen?
Dit was er nog maar één!

2.3.8 De overige tabellen

Categorie is een eenvoudige tabel met slechts 2 velden. Maar welke problemen komen we tegen als we meer velden hebben?
Laten we eens kijken naar de tabel speelgoed. Die heeft 4 velden.
We beginnen met de klasse.
Kijk eens naar de volgende functie voor het toevoegen van nieuw speelgoed:

```
/*-----------------------------------------------------
 * met deze functie kunnen we nieuw speelgoed
 * toevoegen
 -----------------------------------------------------*/
public function toevoegen_speelgoed($speelgoednaam, $categorienummer,
$leenduur_dagen){
  $query = $this->db->prepare(
    "INSERT INTO speelgoed
    (speelgoednaam, categorienummer, leenduur_dagen) VALUES (?,?,?) ");

  $query->bindValue(1, $speelgoednaam);
  $query->bindValue(2, $categorienummer);
  $query->bindValue(3, $leenduur_dagen);

  try{
    $query->execute();
  }catch(PDOException $e){
    die($e->getMessage());
  }
}
```

De functie wordt aangeroepen met 3 parameters (de variabelen tussen de haakjes: `$speelgoednaam, $categorienummer, $leenduur_dagen`).
In de `INSERT` opdracht zie je die 3 parameters weer terugkomen, nu zonder `$`-teken ervoor. Ook bevat de `INSERT` opdracht 3 vraagtekens.
Tenslotte zie je 3 `bindValue` regels met weer die 3 parameters er achter.

Heb je dus 3 parameters, dan moeten al die andere gedeelten ook uit 3 bestaan. Bovendien moeten ze ook nog allemaal in dezelfde volgorde staan.

Hieronder volgt de volledige klasse voor de tabel leeftijdscategorie.

Opdracht 29

Hier de code voor de volledige klasse **speelgoed_class.php**.

- Vervang de code in het bestand **speelgoed_class.php** dat in de map **core/classes** staat.

```php
<?php
class Speelgoed {

  private $db;

  /*-------------------------------------------------
   * De constructor
   -------------------------------------------------*/
  public function __construct($database) {
    $this->db = $database;
  }


  /*-------------------------------------------------
   * met deze functie kunnen we nieuw speelgoed
   * toevoegen
   -------------------------------------------------*/
  public function toevoegen_speelgoed($speelgoednaam, $categorienummer,
$leenduur_dagen){
    $query = $this->db->prepare(
      "INSERT INTO speelgoed
      (speelgoednaam, categorienummer, leenduur_dagen) VALUES (?,?,?) ");

    $query->bindValue(1, $speelgoednaam);
    $query->bindValue(2, $categorienummer);
    $query->bindValue(3, $leenduur_dagen);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }
```

```php
  /*-------------------------------------------------
   * met deze functie kunnen we speelgoed opvragen met
   * behulp van het speelgoednummer
   -------------------------------------------------*/
  public function opvragen_speelgoed($speelgoednummer) {
    $query = $this->db->prepare(
      "SELECT *
      FROM speelgoed
      WHERE speelgoednummer = ?"
    );

    $query->bindValue(1, $speelgoednummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
    return $query->fetchAll();
  }


  /*-------------------------------------------------
   * met deze functie kunnen we speelgoed wijzigen
   -------------------------------------------------*/
  public function wijzigen_speelgoed($speelgoednaam, $categorienummer,
$leenduur_dagen, $speelgoednummer) {
    $query = $this->db->prepare(
      "UPDATE speelgoed SET
      speelgoednaam = ?,
      categorienummer = ?,
      leenduur_dagen = ?
      WHERE speelgoednummer = ?"
    );

    $query->bindValue(1, $speelgoednaam);
    $query->bindValue(2, $categorienummer);
    $query->bindValue(3, $leenduur_dagen);
    $query->bindValue(4, $speelgoednummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }

  /*-------------------------------------------------
   * met deze functie kunnen we speelgoed verwijderen
   -------------------------------------------------*/
```

```php
public function verwijderen_speelgoed($speelgoednummer) {
  $query = $this->db->prepare(
    "DELETE FROM speelgoed
    WHERE speelgoednummer = ?"
   );

  $query->bindValue(1, $speelgoednummer);

  try{
    $query->execute();
  }catch(PDOException $e){
    die($e->getMessage());
  }
}

/*-----------------------------------------------------
 * met deze functie krijgen we een overzicht an al het
 * aanwezige speelgoed
 -----------------------------------------------------*/
public function overzicht_speelgoed() {
  $query = $this->db->prepare(
    "SELECT *
    FROM speelgoed
    ORDER BY speelgoednaam"
  );

  try{
    $query->execute();
  }catch(PDOException $e){
    die($e->getMessage());
  }
  return $query->fetchAll();
}

/*-----------------------------------------------------
 * met deze functie krijgen we een overzicht van het
 * aantal stuks aanwezige speelgoed
 -----------------------------------------------------*/
public function aantal_speelgoed() {
  $query = $this->db->prepare(
    "SELECT COUNT(*)
    FROM speelgoed"
  );

  try{
    $query->execute();
    $rows = $query->fetchColumn();
    return $rows;
  }catch(PDOException $e){
    die($e->getMessage());
```

```
    }
  }

  /*-------------------------------------------------
   * met deze functie kunnen we controleren of een
   * bepaald stuk speelgoed al in de tabel voorkomt
   -------------------------------------------------*/
  public function bestaatAl_speelgoed($speelgoednaam, $categorienummer) {
    $query = $this->db->prepare(
      "SELECT COUNT(*)
      FROM speelgoed
      WHERE speelgoednaam = ?
      AND categorienummer = ?"
    );

    $query->bindValue(1, $speelgoednaam);
    $query->bindValue(2, $categorienummer);

    try{
      $query->execute();
      $rows = $query->fetchColumn();
      if ($rows == 1){
        return true;
      } else {
        return false;
      }
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }


  /*-------------------------------------------------
   * met deze functie kunnen we tijdens het wijzigen
   * controleren of een bepaald speelgoed al in de tabel
   * voorkomt maar hierbij wordt gecontroleerd of het
   * mogelijk al bestaande speelgoed wel een ander
   * speelgoednummer heeft
   -------------------------------------------------*/
  public function bestaatAlMetAnderNummer_speelgoed($speelgoednaam,
$categorienummer, $speelgoednummer) {
    $query = $this->db->prepare(
      "SELECT COUNT(*)
      FROM speelgoed
      WHERE speelgoednaam = ?
      AND categorienummer = ?
      AND speelgoednummer <> ?"
    );

    $query->bindValue(1, $speelgoednaam);
```

```
    $query->bindValue(2, $categorienummer);
    $query->bindValue(3, $speelgoednummer);

    try{
      $query->execute();
      $rows = $query->fetchColumn();
      if ($rows == 1){
        return true;
      } else {
        return false;
      }
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }
}
```

Opdracht 30

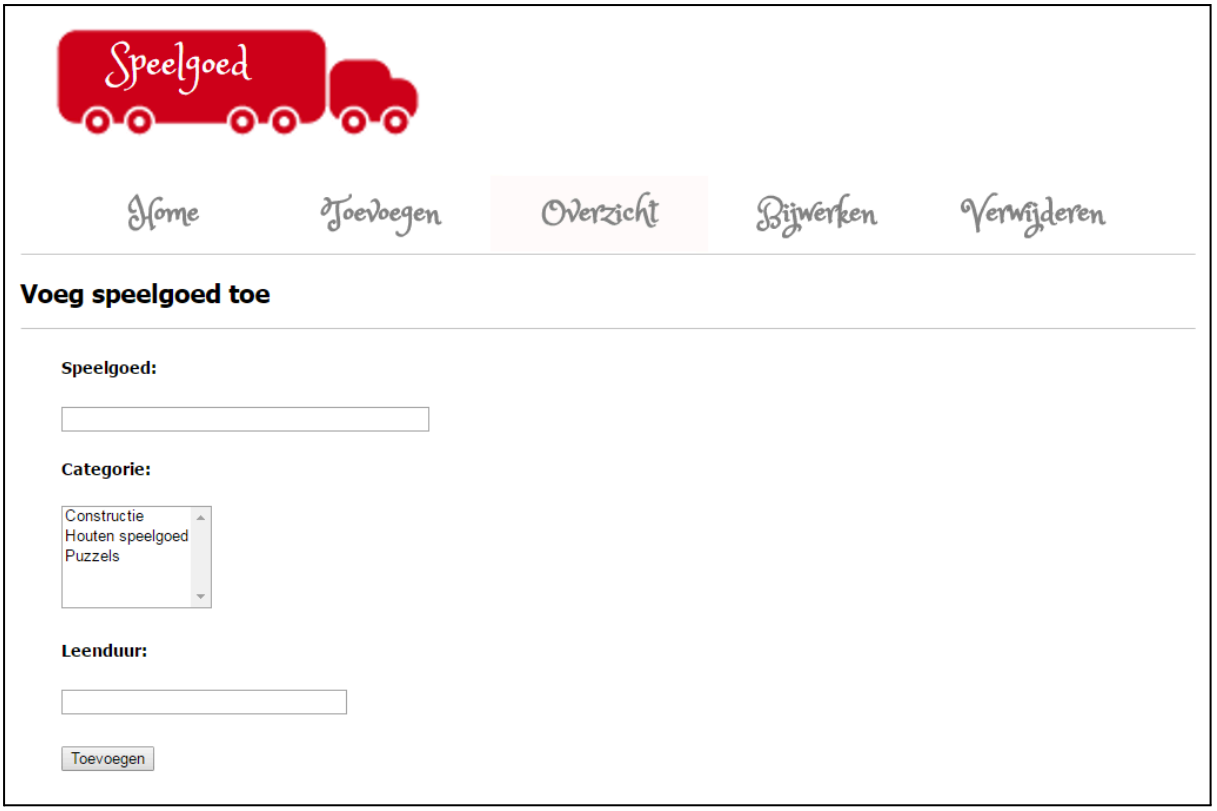
- Waarom heeft de functie **`toevoegen_speelgoed`** maar 3 parameters en **`wijzigen_speelgoed`** 4?

Opdracht 31

- Maak nu ook de klassen **`lid_class.php`**, **`lidsoort_class.php`**, **`uitleen_class.php`** compleet.

2.3.9 Een categorie kiezen bij het speelgoed

Bij het speelgoed sla je het categorienummer op.
Maar echt handig is het niet om op die manier gegevens in te voeren. Beter is het wanneer je uit de reeds ingevoerde categorieën kunt kiezen.
Bijvoorbeeld zo:



Afbeelding 7: Het toevoegen van een categorie aan het speelgoed

Maar dan moet je er wel voor zorgen, dat op basis van de keuze uit de lijst met categorieën, het juiste categorienummer in de tabel speelgoed terecht komt.
In de volgende code zie je hoe je dat kunt doen:

Opdracht 32

Hier een stukje code uit het bestand **speelgoed_toevoegen.php**.

- Vervang het deel van de code waarmee je uit de verschillende categorieën kunt kiezen

```
<form action="" method="post">
  <ul>

    <li>
      <h4>Speelgoed:</h4>
      <input type="text" name="speelgoednaam" size="40">
    </li>

    <li>
      <!-- Kiezen uit de aanwezige categorieën -->
      <h4>Categorie:</h4>
```

```
      <select name="categorienummer" size="5">
        <?php foreach ($categorie->overzicht_categorie() as $rij): ?>
          <option value='<?=$rij['categorienummer']?>'>
            <?=$rij['categorienaam']?>
          </option>
        <?php endforeach; ?>
      </select>
    </li>

    <li>
      <h4>Leenduur:</h4>
      <input type="text" name="leenduur_dagen" size="30">
    </li>

    <li>
      <br>
      <input type="submit" value="Toevoegen">
    </li>
  </ul>
</form>
```

Wat je in deze code ziet is dat de
`<option value='<?=$rij['categorienummer']?>'>`
het categorienummer is en dat hij die waarde doorgeeft via **`$_POST`**.
`<?=$rij['categorienaam']?>` staat tussen de **`option`** tags in en dus is dat de waarde die je op het scherm ziet en waaruit je kunt kiezen.

2.3.10 Een categorie wijzigen bij het speelgoed

Wanneer je bij speelgoed de categorie wilt wijzigen kun je dat op soortgelijke manier doen. Alleen is het dan wel handig dat je kunt zien welke categorie eerder al gekozen werd.
Door het woord **`selected`** toe te voegen in de **`option`** regel kun je dat realiseren. Zoals je in de volgende code kunt zien:

Opdracht 33

Hier een stukje code uit het bestand **`speelgoed_toevoegen.php`**.

- Vervang het deel van de code waarmee je uit de verschillende categorieën kunt kiezen

```
<form action="" method="post">
```

```
  <ul>
    <li>
      <h4>Speelgoed:</h4>
      <input type="text" name="speelgoednaam" size="40"
value='<?=$speelgoednaam?>'>
    </li>

    <li>
      <!-- Kiezen uit de aanwezige categorieën -->
      <h4>Categorie:</h4>
      <select name="catnummer" size="5">
        <?php
        $option = "";
        foreach ($categorie->overzicht_categorie() as $catrij):
          $catnummer = $catrij['categorienummer'];
          $catnaam   = $catrij['categorienaam'];
          if ($catnummer == $categorienummer) {
            $option.="<option value='".$catnummer."'
selected>".$catnaam."</option>";
          } else {
            $option.="<option value='".$catnummer."' >".$catnaam."</option>";
          }
        endforeach;
        echo $option;
        ?>
      </select>
    </li>

    <li>
      <h4>Leenduur:</h4>
      <input type="text" name="leenduur_dagen" size="30"
value='<?=$leenduur_dagen?>'>
    </li>

    <li>
      <br>
      <input type='hidden' name='speelgoednummer' value='<?=$speelgoednummer?>'>
      <input type="submit" value="Wijzigen">
    </li>
  </ul>
</form>
```

Opdracht 34

Je hebt nu voldoende voorbeelden om de volledige CRUD toepassing voor alle tabellen te maken.

- Maak nu de CRUD toepassingen inclusief de menu’s compleet.

3

Uitwerkingen

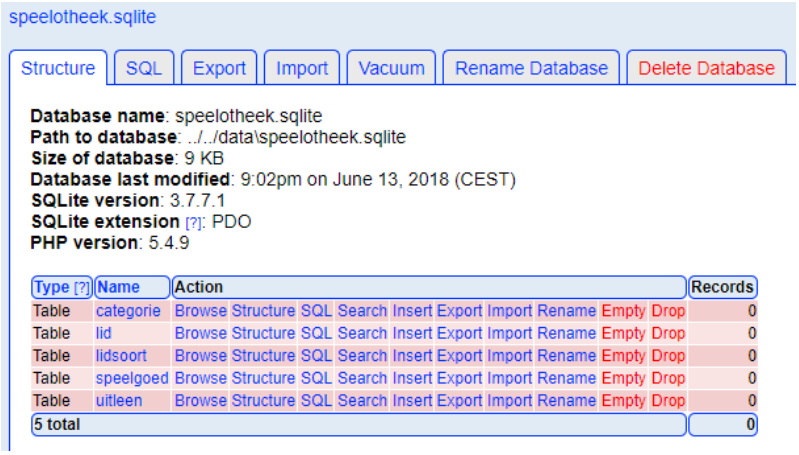
Opdracht 2

```
Grijs
Bruin
Kelder
Jansen
Bergsma
Witte
Dalton
Veenstra
Vergo
Elegast
Spaarzaam
Jongsma
Salamon
Graham
Smit
Klaren
Zandman
```

Opdracht 3

```
<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
    <title>De artikelprijsen van de winkel</title>
  </head>
  <body>
    <?php
      // Maken van verbinding
      $db = new PDO("sqlite:../data/winkel.sqlite");

      // De code hieronder is om eventuele foutmeldingen
      // weer te geven
      $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

      // Dan komt hier de code waarin je gebruik maakt van
      // de database
      $sql = "SELECT artikelnaam, prijs FROM artikelen";
      $resultaat = $db->query($sql);

      foreach($resultaat as $row) {
        echo $row['artikelnaam'].': '.$row['prijs'].'<br>';
      }

      // Sluiten van verbinding
      $db = NULL;
    ?>
  </body>
</html>
```

Opdracht 9



Opdracht 17

speelgoed_class.php

```
<?php
  class Speelgoed {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }
    public function aantal_speelgoed() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM speelgoed"
      );
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }
?>
```

lid_class.php

```
<?php
  class Lid {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }
    public function aantal_lid() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM lid"
      );
```

```
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }
?>
```

lidsoort_class.php

```
<?php
  class Lidsoort {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }
    public function aantal_lidsoort() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM lidsoort"
      );
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }
?>
```

uitleen_class.php

```
<?php
  class Uitleen {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }
    public function aantal_uitleen() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM uitleen"
      );
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }
?>
```

init.php

```
<?php
  require 'connect/verbinden.php';
  require 'classes/categorie_class.php';
  require 'classes/speelgoed_class.php';
  require 'classes/lid_class.php';
  require 'classes/lidsoort_class.php';
  require 'classes/uitleen_class.php';

  $categorie = new Categorie($db);
  $speelgoed = new Speelgoed($db);
  $lid       = new Lid($db);
  $lidsoort  = new Lidsoort($db);
  $uitleen   = new Uitleen($db);

  $errors = array();
?>
```

index.php

```
<?php
require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">

  <head>
    <meta charset="utf-8">
      <link  href='http://fonts.googleapis.com/css?family=Princess+Sofia'  rel='stylesheet'
type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Speel-o-theek</title>
  </head>

  <body>
    <div id="container">
      <img class="center" src="images\speelotheek.png" />
      <?php
        echo '<div>';
        include 'includes/menu.php';
        echo '</div><ul><li>';
        $aantalcategorie = $categorie->aantal_categorie();
        if ($aantalcategorie > 0) {
          echo "Aantal categorieën ".$aantalcategorie;
        } else echo "Er zijn geen categorieën in de database.<br>";
        echo '</li><li>';
        $aantalspeelgoed = $speelgoed->aantal_speelgoed();
        if ($aantalspeelgoed > 0) {
          echo "Aantal speelgoed ".$aantalspeelgoed;
        } else echo "Er is geen speelgoed in de database.<br>";
        echo '</li><li>';
        $aantallid = $lid->aantal_lid();
        if ($aantallid > 0) {
          echo "Aantal leden ".$aantallid;
        } else echo "Er zijn geen leden in de database.<br>";
        echo '</li><li>';
        $aantallidsoort = $lidsoort->aantal_lidsoort();
```

```
        if ($aantallidsoort > 0) {
          echo "Aantal lidsoorten ".$aantallidsoort;
        } else echo "Er zijn geen lidsoorten in de database.<br>";
        echo '</li><li>';
        $aantaluitleen = $uitleen->aantal_uitleen();
        if ($aantaluitleen > 0) {
          echo "Aantal uitleeningen ".$aantaluitleen;
        } else echo "Er zijn geen uitleeningen in de database.<br>";
        echo '</li></ul>';
      ?>
    </div>
  </body>
</html>
```

Opdracht 21

```
<?php
  class Categorie {
    private $db;

    public function __construct($database) {
      $this->db = $database;
    }

    public function aantal_categorie() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM categorie"
      );

      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

    public function toevoegen_categorie($categorienaam){
      $query = $this->db->prepare(
        "INSERT INTO categorie
        (categorienaam) VALUES (?) ");

      $query->bindValue(1, $categorienaam);

      try{
        $query->execute();
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

    public function bestaatAl_categorie($categorienaam) {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
```

```
        FROM categorie
        WHERE categorienaam = ?"
      );

      $query->bindValue(1, $categorienaam);

      try{
        $query->execute();
        $rows = $query->fetchColumn();
        if ($rows > 0){
          return true;
        } else {
          return false;
        }
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

  }
?>
```

Opdracht 23

```
<?php
  class Categorie {
    private $db;

    public function __construct($database) {
      $this->db = $database;
    }

    public function aantal_categorie() {
      $query = $this->db->prepare(
        "SELECT COUNT(*) FROM categorie"
      );

      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

    public function toevoegen_categorie($categorienaam){
      $query = $this->db->prepare(
        "INSERT INTO categorie (categorienaam) VALUES (?) ");

      $query->bindValue(1, $categorienaam);

      try{
        $query->execute();
      }catch(PDOException $e){
        die($e->getMessage());
      }
```

```
    }

    public function bestaatAl_categorie($categorienaam) {
      $query = $this->db->prepare(
        "SELECT COUNT(*) FROM categorie WHERE categorienaam = ?"
      );

      $query->bindValue(1, $categorienaam);

      try{
        $query->execute();
        $rows = $query->fetchColumn();
        if ($rows > 0){
          return true;
        } else {
          return false;
        }
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

    public function overzicht_categorie() {
       $query = $this->db->prepare(
         "SELECT * FROM categorie ORDER BY categorienaam"
       );

       try{
         $query->execute();
       }catch(PDOException $e){
         die($e->getMessage());
       }
       return $query->fetchAll();
    }

  }
?>
```

Opdracht 25

```
<?php
  require 'core/init.php';
?>

<!DOCTYPE html>
<html lang="nl">
  <head>
    <meta charset="utf-8">
      <link  href='http://fonts.googleapis.com/css?family=Princess+Sofia'  rel='stylesheet'
type='text/css'>
    <link rel="stylesheet" type="text/css" href="css/style.css" >
    <title>Categorieën overzicht</title>
  </head>

  <body>
    <div id="container">
```

```
      <?php
        include 'includes/categorie_kop.php';
        include 'includes/categorie_menu.php';
      ?>

      <div id="overzicht">
        <table>
          <thead>
            <tr>
              <th>Categorienummer</th>
              <th>Categorienaam</th>
              <th></th>
              <th></th>
            </tr>
          </thead>

          <tbody>
            <?php
              $resultaat = $categorie->overzicht_categorie();
              foreach ($resultaat as $rij) {
                echo '<tr>';
                echo '
                  <td>'.$rij['categorienaam'].'</td>
                ';
                echo '
                  <td class="muteren">
                  <form action="categorie_wijzigen.php" method="post">
                   <input type="image" src="images/modify.png">
                   <input type="hidden" name="nummer" value="'.$rij['categorienummer'].'">
                  </form>
                </td>
              ';
              echo '
                <td class="muteren">
                  <form action="categorie_verwijderen.php" method="post">
                   <input type="image" src="images/delete.png">
                   <input type="hidden" name="nummer" value="'.$rij['categorienummer'].'">
                  </form>
                </td>
              ';
              echo '</tr>';
              }
            ?>
          </tbody>
        </table>
      </div>
    </div>
  </body>
</html>
```

Opdracht 26

```
<?php
  class Categorie {
    private $db;

    public function __construct($database) {
```

```
  $this->db = $database;
}

public function aantal_categorie() {
  $query = $this->db->prepare(
    "SELECT COUNT(*) FROM categorie"
  );

  try{
    $query->execute();
    $rows = $query->fetchColumn();
    return $rows;
  }catch(PDOException $e){
    die($e->getMessage());
  }
}

public function toevoegen_categorie($categorienaam){
  $query = $this->db->prepare(
    "INSERT INTO categorie (categorienaam) VALUES (?) ");

  $query->bindValue(1, $categorienaam);

  try{
    $query->execute();
  }catch(PDOException $e){
    die($e->getMessage());
  }
}

public function bestaatAl_categorie($categorienaam) {
  $query = $this->db->prepare(
    "SELECT COUNT(*) FROM categorie WHERE categorienaam = ?"
  );

  $query->bindValue(1, $categorienaam);

  try{
    $query->execute();
    $rows = $query->fetchColumn();
    if ($rows > 0){
      return true;
    } else {
      return false;
    }
  }catch(PDOException $e){
    die($e->getMessage());
  }
}

public function overzicht_categorie() {
   $query = $this->db->prepare(
     "SELECT * FROM categorie ORDER BY categorienaam"
   );

   try{
     $query->execute();
   }catch(PDOException $e){
```

```
      die($e->getMessage());
    }
    return $query->fetchAll();
}

public function opvragen_categorie($categorienummer) {
    $query = $this->db->prepare(
      "SELECT * FROM categorie WHERE categorienummer = ?"
    );

    $query->bindValue(1, $categorienummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
    return $query->fetchAll();
}

public function wijzigen_categorie($categorienaam, $categorienummer) {
    $query = $this->db->prepare(
      "UPDATE categorie SET categorienaam = ? WHERE categorienummer = ?"
    );

    $query->bindValue(1, $categorienaam);
    $query->bindValue(2, $categorienummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
}

public function verwijderen_categorie($categorienummer) {
    $query = $this->db->prepare(
      "DELETE FROM categorie WHERE categorienummer = ?"
     );

    $query->bindValue(1, $categorienummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
}

public function bestaatAlMetAnderNummer_categorie($categorienaam, $categorienummer) {
    $query = $this->db->prepare(
      "SELECT COUNT(*) FROM categorie WHERE categorienaam = ? AND categorienummer <> ?"
    );

    $query->bindValue(1, $categorienaam);
    $query->bindValue(2, $categorienummer);

    try{
```

```
        $query->execute();
        $rows = $query->fetchColumn();
        if ($rows > 0){
             return true;
        } else {
             return false;
        }
       }catch(PDOException $e){
        die($e->getMessage());
       }
    }

  }
?>
```

Opdracht 30

Bij het toevoegen van nieuw speelgoed wordt het sleutelveld (speelgoednummer) automatisch gegenereerd en hoeft niet meegegeven te worden.

Bij het wijzigen van bestaand speelgoed moet het sleutelveld (speelgoednummer) wel mee gegeven worden, om het juiste bestand te kunnen opzoeken en wijzigen.

Opdracht 31

lid_class.php

```
<?php
  class Lid {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }
    public function aantal_lid() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM lid"
      );
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }

  public function toevoegen_lid($voornaam, $tussenvoegsel, $achternaam, $adres, $postcode,
$woonplaats, $telefoonnummer, $email, $lidsoortnummer){
    $query = $this->db->prepare(
      "INSERT INTO lid
       (voornaam, tussenvoegsel, achternaam, adres, postcode, woonplaats, telefoonnummer,
email, lidsoortnummer) VALUES (?,?,?,?,?,?,?,?,?) ");

    $query->bindValue(1, $voornaam);
    $query->bindValue(2, $tussenvoegsel);
    $query->bindValue(3, $achternaam);
```

```
    $query->bindValue(4, $adres);
    $query->bindValue(5, $postcode);
    $query->bindValue(6, $woonplaats);
    $query->bindValue(7, $telefoonnummer);
    $query->bindValue(8, $email);
    $query->bindValue(9, $lidsoortnummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }

  public function opvragen_lid($lidnummer) {
    $query = $this->db->prepare(
      "SELECT *
      FROM lid
      WHERE lidnummer = ?"
    );

    $query->bindValue(1, $lidnummer);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
    return $query->fetchAll();
  }

...

?>
```

lidsoort_class.php

```
<?php
  class Lidsoort {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }
    public function aantal_lidsoort() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM lidsoort"
      );
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }
  }

  public function toevoegen_lidsoort($lidsoortnaam){
    $query = $this->db->prepare(
```

```
      "INSERT INTO lidsoort
      (lidsoortnaam) VALUES (?) ");

    $query->bindValue(1, $lidsoortnaam);

    try{
      $query->execute();
    }catch(PDOException $e){
      die($e->getMessage());
    }
  }

...
?>
```

uitleen_class.php

```
<?php
  class Uitleen {
    private $db;
    public function __construct($database) {
      $this->db = $database;
    }

    public function aantal_uitleen() {
      $query = $this->db->prepare(
        "SELECT COUNT(*)
        FROM uitleen"
      );
      try{
        $query->execute();
        $rows = $query->fetchColumn();
        return $rows;
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

      public function toevoegen_uitleen($speelgoednummer, $lidnummer, $datum_uitgeleend,
$datum_terug){
      $query = $this->db->prepare(
        "INSERT INTO uitleen
        (speelgoednummer, lidnummer, datum_uitgeleend, datum_terug) VALUES (?,?,?,?) ");

      $query->bindValue(1, $speelgoednummer);
      $query->bindValue(2, $lidnummer);
      $query->bindValue(3, $datum_uitgeleend);
      $query->bindValue(4, $datum_terug);

      try{
        $query->execute();
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }

    public function opvragen_uitleen($uitleennummer) {
      $query = $this->db->prepare(
        "SELECT *
```

```
        FROM uitleen
        WHERE uitleennummer = ?"
      );

      $query->bindValue(1, $uitleennummer);

      try{
        $query->execute();
      }catch(PDOException $e){
        die($e->getMessage());
      }
      return $query->fetchAll();
    }

      public function wijzigen_uitleen($speelgoednummer, $lidnummer, $datum_uitgeleend,
$datum_terug, $uitleennummer) {
      $query = $this->db->prepare(
        "UPDATE uitleen SET
        speelgoednummer = ?,
        lidnummer = ?,
        datum_uitgeleend = ?
        datum_terug = ?
        WHERE uitleennummer = ?"
      );

      $query->bindValue(1, $speelgoednummer);
      $query->bindValue(2, $lidnummer);
      $query->bindValue(3, $datum_uitgeleend);
      $query->bindValue(4, $datum_terug);
      $query->bindValue(5, $uitleennummer);

      try{
        $query->execute();
      }catch(PDOException $e){
        die($e->getMessage());
      }
    }


   ...

  }
?>
```

Opdracht 32

```
<?php
```

Opdracht 33

```
<?php
```

Opdracht 34

```
<?php
```