El llenguatge PHP (IV): Treball amb BBDDs mitjançant PDO

PDO (https://www.php.net/pdo) és una llibreria oficial orientada a objectes que ve incorporada de sèrie dins de l'intèrpret PHP i que permet als nostres scripts treballar contra múltiples servidors de bases de dades (per connectar-s'hi i fer-hi consultes de tot tipus) d'una única forma homogènia i estàndard. D'aquesta manera, tant se val el SGBD concret amb què treballem (MySQL/MariaDB, PostgreSQL,...la llista completa dels servidors suportats es troba a https://www.php.net/manual/en/pdo.drivers.php) perquè els nostres scripts, si estan programats fent ús de PDO, patiran molts pocs canvis si canviem de SGBD en algun moment.

> NOTA: PDO no és la única llibreria PHP que abstraeix la tasca de connectar i comunicar-se amb diversos SGBDs...n'hi ha altres com "DBA" o "ODBC", però sí que és, amb molta diferència, la més utilitzada. D'altra banda, també existeixen conjunts de funcions (és a dir, construccions no orientades a objectes sinó procedimentals) de baix nivell específiques per cada SBGD concret amb què volguem treballar...molts tutorials d'Internet per exemple expliquen com utilitzar les funcions *mysqli_**, per contactar i treballar amb servidors MySQL/MariaDB (podeu trobar en aquest sentit, com a mostra de documentació rellevant, una referència d'aquestes funcions a https://www.w3schools.com/php/php_ref_mysqli.asp); no obstant, tot i que aquest conjunt de funcions (que, cal dir, també té la corresponent alternativa "OO" en forma de classes i objectes "*mysqli*"!) aporta una funcioalitat prou específica, justament per això, no té la flexibilitat que ofereix PDO

El fet que el mòdul PDO vingui incorporat de sèrie dins de l'intèrpret PHP no fa, no obstant, que automàticament poguem connectar a qualsevol SGBD sense fer res més: per fer-ho haurem d'instal·lar el "driver" específic que serà l'encarregat de "traduïr" les crides genèriques del nostre codi (fetes amb els objectes que proporcionen les classes PDO) a les crides específiques internes realitzades efectivament al SGBD en qüestió (de les quals no ens haurem de preocupar, doncs, en principi, ja que és el "driver" triat qui les fa). Més en concret, per aconseguir que els nostres scripts PHP contactin, per exemple, amb algun servidor MySQL/MariaDB (local o remot, tant és), haurem d'instal·lar, juntament amb l'intèrpret PHP que ja tindrem, els següents paquets addicionals:

*A Ubuntu: el paquet "**php-mysql**". Dins d'aquest paquet s'inclouen les extensions PHP "mysqli" (*/usr/lib/php/xxx/mysqli.so*, amb la corresponent configuració d'activació a */usr/share/phpX.Y-mysql/mysql/mysqli.ini*), "mysqlnd" (*/usr/lib/php/xxx/mysqlnd.so*, amb sa configuració d'activació a */usr/share/phpX.Y-mysql/mysql/mysqlnd.ini*) i "pdo_mysql" (*/usr/lib/php/xxx/pdo_mysql.so*, amb la corresponent configuració d'activació a */usr/share/phpX.Y-mysql/mysql/pdo_mysql.ini*).

> NOTA: L'extensió "mysqlnd" és el "driver nadiu" de baix nivell que serveix com a base per les altres dues, "mysqli" i "pdo_mysql". La diferència entre aquestes darreres és que "mysqli" ofereix un conjunt de funcions (si es fa servir de forma procedimental) o de classes (si es fa servir de forma orientada a objectes) que estan intrínsecament lligades al treball amb servidors MySQL/MariaDB mentre que "pdo_mysql" només ofereix un conjunt de classes (no té variant procedimental) que són genèriques per qualsevol SGBD. Per més informació podeu consultar https://www.php.net/manual/en/mysql.php i també (on es mostren exemples de codi comparatius entre "mysqli" -procedimental i OO- i "pdo_mysql"), https://www.w3schools.com/php/php_mysql_intro.asp

*A Fedora: el paquet "**php-mysqlnd**" (s'instal·larà com a dependència el paquet "php-pdo" si no està instal·lat ja). Dins d'aquest paquet s'inclouen les extensions PHP "mysqli" (*/usr/lib64/php/modules/mysqli.so*, amb la corresponent configuració d'activació a */etc/php.d/30-mysqli.ini*), "mysqlnd" (*/usr/lib64/php/modules/mysqlnd.so*, amb la corresponent configuració d'activació a */etc/php.d/20-mysqlnd.ini*) i "pdo_mysql" (*/usr/lib64/php/modules/pdo_mysql.so*, amb la corresponent configuració d'activació a */etc/php.d/30-pdo_mysql.ini*).

> NOTA: Cal saber que, en tenir instal·lat el subsistema PDO (a diferència del que passa amb el SGBD MySQL/MariaDB -i qualsevol altre-, on, com acabem de dir, cal instal·lar a més un altre paquet addicional per poder treballar-hi) "de regal" ja disposarem de la capacitat de connectar-nos al SGBD SQLite (https://www.sqlite.org), gràcies a la presència dels arxius "pdo_sqlite.so" i "sqlite3.so" (i la seva respectiva configuració, ubicada, per exemple a Fedora, a "/etc/php.d/30-pdo_sqlite.ini" i "/etc/php.d/20-sqlite3.ini"). SQLite és, doncs, l'únic SGBD al qual, sense haver de tenir instal·lat més que l'intèrpret PHP amb el mòdul PDO genèric, podrem contactar.

Un cop instal·lat el paquet necessari (i reiniciat l'intèrpret PHP-FPM), podem confirmar que tot estigui en ordre observant per exemple la sortida de la funció *phpinfo();*: hi haurà d'aparèixer la secció "PDO", on es mostraran els drivers "mysql" i "sqlite" com activats, així com la secció "pdo_mysql", on s'informa, entre d'altres valors, del nom i versió concreta del driver MySQL utilitzat. També hi haurien d'aparèixer les seccions "mysqlnd" (mostrant valors de baix nivell del driver), "mysqli" (mostrant valors corresponents a aquesta extensió particular alternativa a PDO) i, d'altra banda, també la secció "pdo_sqlite".

NOTA: A la secció "pdo_mysql", entre altres valors, es pot veure el de la directiva "pdo_mysql.default_socket", que indica, recordem, la ruta del "socket" emprat per interconnectar localment clients i el servidor SGBD (https://mariadb.com/kb/en/server-system-variables/#socket). Si aquesta directiva -o qualsevol altra d'aquesta secció- es volgués canviar, caldria editar la línia homònima present dins de la secció *[Pdo_mysql]* de l'arxiu "/etc/php.ini"

Exemple de codi: connexió al SGBD

El següent codi (que pot executar-se tant des del terminal amb la comanda *php* o bé, si està ubicat dins d'un "DocumentRoot" d'algun servidor web, des del propi navegador), mostra com realitzar una connexió a una determinada base de dades (anomenada en aquest cas "midb") allotjada en un servidor MariaDB (funcionant en aquest cas sobre la mateixa màquina on estem executant l'intèrpret PHP, tal com indica la IP 127.0.0.1) utilitzant l'usuari "root" amb una contrasenya tal com "1234":

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e){
  echo "Connection failed: " . $e->getMessage();
}
$conn = null;
?>
```

La connexió a qualsevol SGBD mitjançant el subsistema PDO es realitza sempre de la mateixa manera: fent servir el constructor *PDO()*. Aquest constructor pot tenir diferents paràmetres segons el SGBD particular amb què volem connectar-nos (més sobre això més endavant) però en tot cas, retorna un objecte (que hem anomenat *$conn*) que representa la connexió establerta amb el SGBD triat, a partir de la qual podrem començar a treballar.

De fet, un cop creat l'objecte *$conn*, el primer que fa el codi anterior és establir certes característiques ("atributs") a la connexió corresponent mitjançant el mètode *setAttribute()*, el qual té dos paràmetres: el nom de l'atribut en qüestió i el valor a establir-li (si es vol establir més d'un atribut, caldrà escriure una línia *setAtribute()* per cadascun d'ells). Aquest pas és opcional, però sempre convé conéixer alguna d'aquests atributs que tenim disponibles per valorar si ens interessaria especificar-los. La llista d'atributs genèrics (perquè després cada "driver" en pot afegir algun de propi) es troba aquí https://www.php.net/manual/es/pdo.setattribute.php però a continuació es mostra una taula dels més comuns:

*Atribut *PDO::ATTR_ERRMODE* : El seu valor *PDO::ERRMODE_EXCEPTION* fa que qualsevol error que es produeixi durant la connexió o qualsevol altra operació amb la base de dades es llanci com a excepció (la qual es pot capturar en el bloc *catch* següent, veieu nota següent)

NOTA: Com es pot veure, tot el codi anterior està dins d'un bloc *try/catch* per tal d'assegurar-se que si la connexió al SGBD no es realitza (en aquest cas, si l'objecte *$conn* no és creat correctament, ja sigui perquè el SGBD no respon, perquè els paràmetres introduïts al constructor no són correctes, etc), es generi una excepció de tipus "PDOException" i, en aquest cas, es mostri, com a conseqüència, el missatge d'error pertinent i no es faci res més. El que cal tenir clar és que l'excepció "PDOException" només es generarà si hem especificat el valor PDO::ERRMODE_EXCEPTION a l'atribut PDO::ATTR_ERRMODE; si no es fa això, es dispararà un error "estàndard" però no pas una excepció.

*Atribut *PDO::ATTR_TIMEOUT* : El seu valor ha de ser un número sencer que indica la quantitat de segons que PDO esperarà en rebre resposta del SGBD abans de deixar-ho córrer (i, en aquest cas, disparar una excepció). El valor per defecte depèn del driver natiu que es faci servir.

*Atribut *PDO::ATTR_DEFAULT_FETCH_MODE* : El seu valor indica el mode "fetch" que es farà servir en l'obtenció de dades rebudes d'una consulta. La descripció d'aquests modes "fetch" està disponible a https://www.php.net/manual/es/pdostatement.fetch.php i en podem destacar *PDO::FETCH_NUM*, *PDO::FETCH_ASSOC* o *PDO::FETCH_BOTH*, entre altres. En veurem el seu significat i la seva utilitat més endavant

NOTA: Els atributs de la connexió es poden indicar d'una forma alternativa, en lloc de fer servir el mètode *setAttribute()* Concretamen, es poden indicar com a quart paràmetre del contructor *PDO()* en forma d'array associatiu (per ubicar si n'hi hagués més d'un atribut a establir). A continuació es mostren un parell d'exemples equivalents a l'anterior; el primer aquí...

```
<?php
$options = [ PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION ];
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234", $options);
//...
?>
```

...i el segon (que és el mateix però amb una altra sintaxi), aquí:

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234",
                                  array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
//...
?>
```

A continuació, es mostra un missatge de "Connected successfully" si s'ha establert la connexió amb èxit, (o, com hem dit, un missatge d'error amb el missatge d'excepció si s'ha produït algun problema). Finalment, destruint l'objecte *$conn* (és a dir, reassignant-li el valor *null*) tanquem la connexió amb la base de dades (i, així, alliberem l'estructura de la memòria). Cal dir que, aquesta destrucció normalment no es realitzarà tan aviat, ja que el codi d'aquest exemple només escriu un missatge i prou...en codis més realistes caldrà realitzar una o més consultes a la base de dades i, per tant, la destrucció de *$conn* només s'haurà de realitzar quan totes aquestes operacions contra el SGBD hagin finalitzat (tot i que, en realitat, si no s'anul·la explícitament l'objecte *$conn*, l'intèrpret PHP el destruirà automàticament quan l'script que l'hagi creat finalitzi la seva execució...sempre que no s'estiguin utilitzant, això sí, conexions persistents, veieu https://www.php.net/manual/en/pdo.connections.php).

El més interessant del codi anterior està en observar què, a excepció de la part marcada en color lila (part que té un nom: "DSN", de "Database Source Name"), tota la resta del codi serà exactament igual ja ens estiguem connectant a un SGBD MySQL/MariaDB com a un SQLite, o a un PostgreSQL, o a un MSSQLServer, o a un OracleSQLServer, etc, etc. És a dir, canviant simplement els paràmetres de connexió, tota la resta de la lògica que haguem implementat en el nostre codi PHP no hauria de canviar pas, possibilitant així, si fos necessari, una migració de SGBDs molt senzilla.

NOTA: A continuació es llista la sintaxi dels DSN que farem servir més sovint:

*__MySQL/MariaDB via TCP/IP__ (documentada a https://www.php.net/manual/es/ref.pdo-mysql.connection.php):
"mysql:host=nom.oIP.del.servidor;port=3306;dbname=nomBD"
+El port només cal indicar-lo si fos un valor diferent del port per defecte (3306).
+El nom de la base de dades no caldrà indicar-lo si a la connexió es volen realitzar operacions genèriques contra el servidor no circumscrites a una BD en concret (per exemple, per veure les bases de dades existents, o crear-ne una de nova, etc)
+Atenció perquè si s'assigna el valor "localhost" a l'opció "host", automàticament l'intèrpret PHP utilitzarà el mecanisme de connexió "unix socket" (descrit a continuació), amb els problemes que això comporta; és per això que cal indicar la IP 127.0.0.1 si es vol utilitzar el mecanisme TCP/IP.
+És recomanable indicar també el valor *charset=utf8mb4* dins del DSN per assegurar-se que el joc de caràcters utilitzat en els missatges enviats des de l'script PHP al servidor MySQL estan codificats convenientment

*__MySQL/MariaDB via "unix socket"__ (documentada al mateix lloc):
"mysql:unix_socket="/ruta/arxiu.sock";dbname=nomBD"
+A Fedora, per exemple,la ruta de l'"arxiu.sock" és "/var/lib/mysql/mysql.sock", tal com ja s'ha vist en documents anteriors.
+En aquest cas, el valor del paràmetre "contrasenya" del contructor *PDO()* és irrellevant.
+Cal tenir en compte, però, que aquesta possibilitat només funcionarà si el SGBD està funcionant en la màquina local i si l'script PHP s'executa amb permisos de "root" del sistema (o d'un altre usuari que tingui activada l'autenticació via aquest mecanisme de connexió, cosa que per defecte l'usuari "www-data"/"apache", responsable dels pools de PHP-FPM no té!).
+És recomanable indicar també el valor *charset=utf8mb4* dins del DSN per assegurar-se que el joc de caràcters utilitzat en els missatges enviats des de l'script PHP al servidor MySQL estan codificats convenientment

*__SQLite__ (documentada a https://www.php.net/manual/es/ref.pdo-sqlite.connection.php):
"sqlite:/ruta/absoluta/fitxer.sqlite"
+També es podria utilitzar com a base de dades una de tipus temporal que desapareixerà en tancar la connexió; per fer això, només caldria indicar "*sqlite:*" com a DSN
+Els valors dels paràmetres "usuari" i "contrasenya" del contructor *PDO()* seran *null* (perquè no tenen semàntica)

*__PostgreSQL__ (documentada a https://www.php.net/manual/es/ref.pdo-pgsql.connection.php):
"pgsql:host=nom.oIP.del.servidor;port=3306;dbname=nomBD;user=nomUsuari;password=contrasenya"
+Les opcions "user=" i "password=" del DSN són opcionals, però si s'especifiquen, llavors els valors dels paràmetres "usuari" i "contrasenya" del contructor *PDO()* no es tindran en compte

Exemple de codi: execució d'una sentència SQL

El següent codi mostra com executar una sentència SQL qualsevol contra el SGBD indicat (sempre que tinguem els permisos adients) de la qual no n'esperem obtenir dades (és a dir, sentències com *CREATE DATABASE*, *CREATE TABLE*, *INSERT*, *UPDATE*, *DELETE*, etc). El punt clau aquí radica en el mètode *exec()*; pertanyent a l'objecte *$conn*, el qual serveix per llençar contra el SGBD la sentència SQL indicada com a paràmetre.

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql= "CREATE DATABASE midb";
  $conn->exec($sql);
  echo "Database created successfully";
} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

NOTA: Fixeu-vos com que, en aquest cas, com s'està executant una sentència SQL independent de qualsevol base de dades concreta, al DSN no s'ha especificat el valor *dbname=* Normalment no serà el cas.
NOTA: Fixeu-vos també com no cal afegir el ";" final a les sentències SQL. D'altra banda, podríem haver escrit la sentència SQL directament com a valor del paràmetre del mètode *exec()* però hem preferit fer-ho a través d'una variable.
NOTA: El mètode *exec()* retorna el nombre de files afectades per la sentència SQL executada o *false* si hi ha algun error

Amb el mateix codi de l'exemple anterior es podrien crear totes les taules necessàries fent servir en aquest cas la sentència *CREATE TABLE* pertinent (i indicant, ara sí, la base de dades de treball a l'opció *dbnane=* del DSN!). De fet, de cara als propers exemples que estudiarem en els següents apartats, suposarem que tenim creada dins de la base de dades "midb" una taula anomenada "MyGuests" amb la següent estructura:

```
CREATE TABLE MyGuests (
id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

NOTA: La columna "id" és numèrica però de tipus *AUTO_INCREMENT* Això vol dir que en cada inserció nova no caldrà indicar-li cap valor concret perquè el propi SGBD li assignarà el valor sencer que hi hagi lliure de forma consecutiva al darrer ja existent
NOTA: La columna "reg_date" és de tipus *TIMESTAMP* amb valor per defecte *CURRENT_TIMESTAMP*. Això vol dir que en cada inserció nova no caldrà indicar cap valor concret perquè el propi SGBD li assignarà el valor de la data i hora actual de la inserció.La sentència ON UPDATE CURRENT_TIMESTAMP serveix per garantir que aquesta data i hora s'actualitzi cada cop que alguna de les altres columnes del registre en qüestió s'actualitzi via UPDATE (per defecte, això no es fa).

Una manera alternativa d'executar sentències SQL és, en lloc d'haver-les d'indicar incrustades dins del codi PHP com hem vist, tenir-les escrites en un fitxer ".sql" i llavors obligar al nostre script a llegir el contingut d'aquest fitxer per tal que, seguidament, amb el mateix mètode *exec()*, les executi d'igual manera, una darrera l'altra. D'aquesta forma, ens podem estalviar d'incloure llargues (o múltiples) sentències SQL (per exemple, quan es volen crear múltiples taules) entremig de codi PHP i, per tant, podrem tenir-ho tot més ben endreçat. Posarem en pràctica aquesta tècnica per crear la taula "MyGuests" l'estructura de la qual acabem d'especificar; l'únic que haurem de fer és escriure la sentència CREATE TABLE tal es mostra en aquest document dins d'un fitxer de text (que anomenarem "taula.sql") que guardarem a la mateixa carpeta on es troba el nostre script PHP, el qual ara haurà de ser aquest:

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql= file_get_contents(__DIR__ . "/taula.sql");
  $conn->exec($sql);
  echo "Table created successfully";
} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

Com es pot veure, aquí la clau està en la funció predefinida de PHP *file_get_contents()*, la qual retorna el text tal com està contingut en el fitxer la ruta del qual s'ha indicat com a paràmetre (o *false* si hi ha hagut algun error).

> NOTA: No cal que el fitxer "sql" estigui ubicat a la mateixa carpeta que l'script PHP però allà on sigui caldrà indicar-ho amb la ruta pertinent dins de *file_get_contents()* i l'script PHP hi haurà de tenir permisos de lectura. Si no es vol que els fitxers "sql" estiguin disponibles pels visitants a pesar d'estar ubicats dins d'un "DocumentRoot", es pot afegir la següent configuració al "virtualhost" en qüestió de l'Apache (o al del propi servidor global):
>
> ```
> <Files ~ "\.sql$">
>  Require all denied
> </Files>
> ```

Exemple de codi: execució de transaccions

Un cop ja tenim la base de dades creada i les taules necessàries també, el següent pas serà omplir-les de dades (fent servir la sentència INSERT). Per fer això, com ja sabem, només ens caldrà tornar a utilitzar el mètode *exec()*. Un exemple senzill seria el següent:

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql= "INSERT INTO MyGuests (firstname,lastname,email) VALUES ('John','Doe','john@example.com')";
  $conn->exec($sql);
  echo "New record created successfully";
} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

No obstant, si volem realitzar múltiples insercions (o, en general, múltiples accions sobre una base de dades), la majoria de cops ens voldrem assegurar que es realitzin totes de forma completa sense excepció i, si no pogués ser perquè hi hagués algun error en algun moment donat durant la realització d'aquestes múltiples accions, llavors ens voldrem assegurar de no quedar-nos "a la meitat" sinó revertir el procés per a tornar a l'estat inicial de la base de dades "sense que no hagi passat res". És a dir, voldrem realitzar transaccions indivisibles per evitar deixar la base de dades en estats intermitjos inestables.

La manera de realitzar transaccions amb PDO és mitjançant els mètodes *beginTransaccion(), exec()* i *commit()/rollback()* de l'objecte *$conn*. A diferència dels exemples anteriors, ara tot mètode *exec()* -executat, això sí, després del mètode *beginTransaction()* - no realitza al moment la sentència SQL associada sinó que només la "guarda", a l'espera que es dispari tota la transacció al complet (formada per totes les sentències SQL acumulades pels diversos mètodes *exec()* indicats) mitjançant el mètode *commit()*, mètode que serà qui efectivament executarà totes les sentències indicades als múltiples *exec()* anteriors. Si en aquest moment concret de fer el "commit" aparegués algun error, es dispararà l'excepció pertinent, moment que aprofitem per tirar enrera tota la transacció mitjançant el mètode *rollback()*. A continuació es veu un exemple utilitzant, en aquest cas, vàries sentències INSERT dins d'una transacció.

```php
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $conn->beginTransaction();
  $conn->exec("INSERT INTO MyGuests (firstname,lastname,email) VALUES ('Jack','Poe','jack@example.com')");
  $conn->exec("INSERT INTO MyGuests (firstname,lastname,email) VALUES ('Mary','Moe','mary@example.com')");
  $conn->commit();
  echo "New records created successfully";
} catch(PDOException $e){
  $conn->rollback();
  echo $e->getMessage();
}
$conn = null;
?>
```

Exemple de codi: execució de "prepared statements"

Un "prepared statement" ("instrucció preparada") és una tècnica que s'utilitza per executar les mateixes sentències SQL (o similars) repetidament amb alta eficiència. Funciona bàsicament en tres fases:

1.-Preparació:: es crea una "plantilla" de la sentència SQL a executar, on es marquen alguns elements d'aquesta amb un nom determinat (precedit del símbol ":") però sense cap valor concret assignat; seran els "forats" que s'ompliran de valors concrets en el moment oportú. Per exemple: *INSERT INTO MyGuests (firstname,lastname,email) VALUES (:nom, :cognom, :correu);* Aquesta plantilla s'envia en aquesta fase al SGBD per tal que aquest l'analitzi, compili i optimitzi i així la tingui preparada per quan s'invoqui de forma efectiva.

NOTA: Existeix una forma alternativa d'indicar els "forats" dins de la plantilla; en lloc de fer servir la sintaxi ":nom", es pot fer indicar simplement un interrogant (així, per exemple: *INSERT INTO MyGuests (firstname,lastname,email) VALUES (?, ?, ?);* No obstant, fent-ho així els "forats" seran anònims en el sentit que no tenen nom, i això pot dificultar el seu processament més endavant.

2.-Enllaçat: S'associa cadascun dels "forats" amb nom anteriors (anomenats paràmetres) amb un determinat valor concret (obtingut de qualsevol manera: o bé assignat directament, o bé recollit d'alguna interacció d'un usuari -com podria ser un formulari web-, etc)

3.-Execució: Es dispara l'execució de la plantilla preparada de la sentència SQL (la "instrucció") fent servir els valors concrets assignats als paràmetres en el pas anterior. A partir d'aquí, es pot executar la mateixa instrucció tantes vegades com vulgui amb diferents valors pels paràmetres definits sense apenes penalització de rendiment.

En comparació amb l'execució directa de sentències SQL, els "prepared statements" tenen aquests avantatges principals:

*Les declaracions preparades redueixen el temps d'anàlisi, ja que la preparació de la consulta només es fa una vegada (tot i que la instrucció s'executi diverses vegades)

*Els paràmetres vinculats minimitzen l'amplada de banda al servidor, ja que només cal enviar els valors associats cada vegada i no tota la consulta.

*Les declaracions preparades són molt útils contra les injeccions SQL perquè els valors dels paràmetres, que es transmeten més tard mitjançant un protocol diferent, no cal escapar-los expressament: si la plantilla de declaració original no es deriva d'una entrada externa, no es pot produir la injecció SQL

Aquest darrer punt és especialment important i és el motiu principal pel qual sempre es recomana fer servir "prepared statements" per qualsevol sentència (INSERTs, UPDATEs, etc) que rebi valors procedents de l'exterior (el més habitual és via formularis web) ja que així estarem protegits enfront possibles valors maliciosos que podrien introduïr-se dins del SQL a executar i que podrien derivar en una pèrdua de dades (en forma de robatori o d'eliminació) o fins i tot del control del SGBD sencer.

El següent codi mostra com executar un INSERT (també podria haver sigut un UPDATE, un DELETE, etc) però, a diferència dels exemples anterior, ho fa mitjançant un "prepared statement", assegurant- se així que els valors a introduïr en la base de dades estiguin sanititzats. Per aconseguir-ho, farem servir el mètode *prepare()* de l'objecte *$conn* , el qual retorna un altre tipus d'objecte (l'anomenarem *$stmt*), que representa la instrucció preparada i que conté dos mètodes que farem servir: *bindParam()* -que serveix per vincular cada paràmetre de la instrucció amb un valor concret, que en aquest cas serà el que tingui en el moment de la seva execució una determinada variable) i *execute()* , responsable de l'execució efectiva de la instrucció amb els valors ja vinculats.

> NOTA: Existeix un mètode similar a *bindParam()* anomenat *bindValue()* que en lloc de vincular un paràmetre de la plantilla amb una variable (a la qual es consultarà el seu valor concret en el moment de fer l'*execute()*) com fa *bindParam()*, vincula el paràmetre directament a un valor concret i fixe.

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname,lastname,email) VALUES (:nom,:cognom,:correu)");
  //El tercer paràmetre del mètode bindParam() força a que el valor en qüestió sigui d'un tipus determinat (o error)
  //Podeu veure totes les constants existents a https://www.php.net/manual/es/pdo.constants.php
  $stmt->bindParam(":nom", $unavar, PDO::PARAM_STR);
  $stmt->bindParam(":cognom", $otravar, PDO::PARAM_STR);
  $stmt->bindParam(":correu", $otramas, PDO::PARAM_STR);
  //Omplo de valors els paràmetres vinculats i realitzo una inserció
  $unavar= "Terry";
  $otravar= "Smith";
  $otramas= "terry@example.com";
  $stmt->execute();
  //Omplo de nou els paràmetres vinculats amb altres valors i realitzo una nova inserció
  $unavar= "Anne";
  $otravar= "Goodman";
  $otramas= "anne@example.com";
  $stmt->execute();

  echo "New records created successfully";
} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

> NOTA: Si no hi ha valors externs per omplir la plantilla de la instrucció (és a dir, si la plantilla en realitat és una sentència SQL tal qual, sense "forats") també podem fer servir aquest mecanisme dels "prepared statements", només que sense haver d'utilitzar el mètode *bindParam()* per res; només el *prepare()* i, quan ho necessitem, el *execute()*.
> NOTA: En el cas d'haver preparat la instrucció indicant els seus paràmetres amb interrogants en lloc de amb noms, al mètode *bindParam()* llavors haurem de fer servir nombres (començant per l'1!) per indicar cadascun dels valors que els volem associar, en ordre (és a dir, per exemple, en l'exemple anterior, el valor del nom l'hauríem d'assignar així: *$stmt->bindParam(**1**, $unavar, PDO::PARAM_STR);* i el del cognom així, per exemple: *$stmt->bindParam(**2**, $unavar, PDO::PARAM_STR);*)

Una sintaxi alternativa equivalent a l'exemple anterior seria prescindir del mètode *bindParam()* i llavors fer la vinculació dels valors concrets amb els paràmetres de la plantilla directament en el mètode *execute()*, fent servir per això el seu primer paràmetre, que serà un array (de tipus associatiu si hem fet servir paràmetres amb nom així *":nomParametre"* o bé de tipus indexat si hem fet servir paràmetres sense nom mitjançant interrogants) cada element del qual tindrà el nom d'un paràmetre i el valor que li volguem assignar. És a dir:

```php
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname,lastname,email) VALUES (:nom,:cognom,:correu)");
  $stmt->execute(array(":nom"=>"Tommy",":cognom"=>"Harrison",":correu"=>"tommy@example.com"));
  $stmt->execute(array(":nom"=>"Lisa",":cognom"=>"Burroughs",":correu"=>"lisa@example.com"));
  echo "New records created successfully";
} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

NOTA: L'objecte *$stmt* té un altre mètode que ens podrà ser d'utilitat en algun moment, *rowCount()*, el qual no té paràmetres i retorna el nombre de files afectades per la darrera sentència DELETE, INSERT o UPDATE executada per l'objecte *$stmt* en qüestió, o el nombre de files retornades per una sentència SELECT o SHOW

Exemple de codi: obtenció de dades d'una consulta, i el seu recorregut

Fins ara, cap de les sentències que hem executat en els codis anteriors retornaven cap dada. En aquest apartat veurem les diferents formes d'obtenir i mostrar la (meta)informació emmagatzemada en una base de dades a partir de fer consultes SELECT, SHOW o similars. Per començar, la consulta SQL ja no es realitzarà mitjançant el mètode *exec()* sinó amb el mètode ***query()***, el qual retornarà el resultat en forma d'objecte que anomenarem *$res* i que representa el conjunt de registres (files) obtinguts (els quals caldrà recòrrer un a un de forma seqüencial per anar processant-los). Per fer aquest recorregut de les files que formen el resultat obtingut per *query()*, aquest objecte-conjunt de dades *$res* ofereix diversos mètodes, entre els quals podem destacar els següents:

**El mètode* *fetchObject() :* Cada cop que s'executa, retorna la següent fila present al conjunt, i la retorna com un objecte PHP on el nom de les seves propietats són els noms de les columnes presents a la fila i els valors de les seves propietats són els valors d'aquestes columnes respectives. Si no n´hi ha més files a retornar (perquè s'ha arribat al final, retornarà *false*).

**El mètode* *fetchColumn() :* Cada cop que s'executa, retorna el valor d'una determinada columna (la posició de la qual s'ha d'indicar com a valor numèric -començant per 0- en el seu 1r paràmetre) de la següent fila present al conjunt. Si no n´hi ha més files (perquè s'ha arribat al final, retornarà *false*).

**El mètode* *fetch() :* Cada cop que s'executa, retorna la següent fila present al conjunt, i la retorna d'una forma diferent segons el valor que tingui el seu primer paràmetre. Si no n´hi ha més files a retornar (perquè s'ha arribat al final, retornarà *false*). Podem destacar, d'entre tots els possibles (veieu https://www.php.net/manual/es/pdostatement.fetch.php), els següents valors:

* *PDO::FETCH_ASSOC*: La fila és retornada en forma d'array associatiu on les claus són els noms de les columnes de la fila obtinguda i els valors són els corresponents valors d'aquestes columnes

* *PDO::FETCH_NUM*: La fila és retornada en forma d'array on l'índex numèric (començant pel 0) representa la posició de cada columna de la fila obtinguda i el valor corresponent és el del valor de la columna en qüestió

* *PDO::FETCH_BOTH* (valor per defecte si no s'indica): La fila és retornada en forma d'array associatiu i en forma d'array indexat (essent una combinació de *PDO::FETCH_NUM* i de *PDO::FETCH_ASSOC* , doncs)

NOTA: Es pot canviar el valor per defecte del primer paràmetre del mètode *fetch()* per a què no sigui *PDO::FETCH_BOTH* si abans assignem el valor desitjat mitjançant el mètode *setAttribute()* a l'atribut PDO::ATTR_DEFAULT_FETCH_MODE de l'objecte *$conn*, tal com ja es va comentar, o alternativament, amb el mètode *setFetchMode()* de l'objecte *$res*.

* *PDO::FETCH_OBJ*: La fila és retornada en forma d'objecte PHP on el nom de les seves propietats són els noms de les columnes presents a la fila i els valors de les seves propietats són els valors d'aquestes columnes respectives

NOTA: *El mètode fetchObject(), tal com es veu, és equivalent al mètode fetch(PDO::FETCH_OBJ)*

* *PDO::FETCH_BOUND:* Aquest mode va associat a l'ús del mètode *bindColumn()* de *$res* El que fa és assignar cada valor individual obtingut de la consulta SQL (indicat com a primer paràmetre de *bindColumn()*, ja sigui mitjançant el nom de la columna al que correspon o bé la seva posició numèrica començant per 1) a una variable independent. D'aquesta manera, no tenim arrays ni objectes que recórrer, tant sols valors individuals per cada columna de la fila actual.

NOTA: L'objecte *$res*, en realitat és del mateix tipus que l'objecte *$stmt* vist a l'apartat de "prepared statements". Això significa que podem accedir al conjunt de resultats obtinguts per una consulta si aquesta es fa no mitjançant query() sinó mitjançant la combinació prepare()/execute(), així:

```
$stmt = $conn->prepare("SELECT * from MyGuests");
$stmt->execute();    //El propi objecte $stmt contindrà el resultat obtingut de l'execució de la consulta!
$fila=$stmt->fetch(PDO::FETCH_ASSOC);
```

El següent codi mostra un exemple d'execució d'una consulta SQL de la qual s'esperen obtenir dades, dades que un cop obtingudes es recorren de diferents maneres per anar processant-les (en aquest cas només per mostrar-les en format de taula HTML i prou) registre a registre (o dit d'una altra forma, "línia" a "línia"). Per provar-lo, s'està suposant que dins de la base de dades "midb" ja existeix la taula "MyGuests" amb diversos registres ja introduïts.

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $res = $conn->query("SELECT * FROM MyGuests");
  //Recórrer el resultat en forma d'array associatiu
  echo "<table>";
  while ($fila=$res->fetch(PDO::FETCH_ASSOC)){
    echo "<tr>";
    //Es podria mostrar el valor d'una columna individual (per exemple, "firstname") amb la sintaxi $fila["firstname"]
    foreach ($fila as $clau => $valor){
        echo "<td>" . $valor . "</td>";
    }
    echo "</tr>";
  }
  echo "</table><hr>";

//-------------------------------------------------------------------------------------------------------

  $res = $conn->query("SELECT * FROM MyGuests");
  //Recórrer el resultat en forma d'array indexat
  echo "<table>";
  while ($fila=$res->fetch(PDO::FETCH_NUM)){
    echo "<tr>";
    //Es podria mostrar el valor d'una columna individual (per exemple, "firstname") amb la sintaxi $fila[0]
    foreach ($fila as $valor){
        echo "<td>" . $valor . "</td>";
    }
    echo "</tr>";
  }
  echo "</table><hr>";

//-------------------------------------------------------------------------------------------------------
```

```
  $res = $conn->query("SELECT * FROM MyGuests");
  //Recórrer el resultat en forma d'objecte
  echo "<table>";
  while ($fila=$res->fetch(PDO::FETCH_OBJ)){
    echo "<tr><td>" . $fila->firstname . "</td><td>" . $fila->lastname . "</td><td>" . $fila->email . "</td></tr>";
  }
  echo "</table><hr>";

//-------------------------------------------------------------------------------------------------------

  $res = $conn->query("SELECT firstname,lastname FROM MyGuests");
  //S'assignarà el valor de la columna "firstname" obtingut a la variable $nom
  $res->bindColumn("firstname", $nom);
  //S'assignarà el valor de la 2ª columna ("lastname") obtingut a la variable $cognom
  $res->bindColumn(2, $cognom);
  echo "<table>";
  while ($fila=$res->fetch(PDO::FETCH_BOUND)){
      echo "<tr><td>" . $nom . "</td><td>" . $cognom . "</td></tr>";
  }
  echo "</table>";


} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

NOTA: El codi anterior es pot provar amb altres tipus de consultes, com per exemple *SHOW DATABASES* o altres...el funcionament serà igual

A continuació mostrem una variant del codi anterior on no es mostra tot el contingut de la taula "MyGuests" sinó només el registre corresponent a la persona amb ID nº3. La idea darrera aquest exemple és que el valor d'aquest camp ID hauria de ser hipotèticament dinàmic perquè s'hauria obtingut a partir d'un formulari web o similar, amb la qual cosa aquesta seria una consulta on intervindria un valor extern. Per tant, l'haurem de realitzar fent servir una instrucció preparada (un "prepared statement") per així intentar evitar possibles atacs d'injecció SQL.

NOTA: És a dir, la idea és evitar escriure una consulta com la següent, que és molt vulnerable a la introducció de valors maliciosos: *$res= $conn->query("SELECT firstname FROM MyGuests* ***WHERE id = " . $_GET["id"]);*** Imagineu per exemple que un usuari indica el valor "1%3BDELETE+FROM+MyGuests" : això esborraria tota la taula!

```
<?php
try {
  $conn = new PDO("mysql:host=127.0.0.1;dbname=midb", "root", "1234");
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT * FROM MyGuests WHERE id=:paramid");
  $stmt->bindParam(":paramid", $valorrecollit, PDO::PARAM_INT);
  //En lloc d'aquest valor fixe (3) la idea és que aquest valor fos recollit dinàmicament via $_GET["id"] o similar
  $valorrecollit=3;
  $stmt->execute();
  //Recordem, l'objecte $stmt allotja també el resultat obtingut (en aquest cas, una sola fila)
  $fila=$stmt->fetch(PDO::FETCH_ASSOC);
  foreach ($fila as $clau => $valor){
    echo "<b>" . $clau . "</b>:" . $valor . "</br>";
  }
} catch(PDOException $e){  echo $e->getMessage();  }
$conn = null;
?>
```

NOTA: L'objecte *$stmt* té un altre mètode que ens podrà ser d'utilitat en algun moment, *columnCount()*, el qual no té paràmetres i retorna el nombre de columnes retornades per la consulta executada per l'objecte *$stmt* en qüestió
NOTA: Teniu tota la referència dels mètodes i propietats pertanyents a l'objecte $conn (generat pel constructor *PDO()*) i a l'objecte $stmt (generat de diverses formes a partir de $conn) a la documentació oficial, respectivament https://www.php.net/manual/es/class.pdo.php i https://www.php.net/manual/es/class.pdostatement.php

EXERCICIS:

1.-a) Arrenca una màquina virtual amb la seva tarja en mode "adaptador pont" i on tinguis funcionant tant un servidor Apache+PHP-FPM com un servidor MariaDB. Tot seguit, crea-hi un arxiu anomenat, per exemple, "init.sql" dins de la seva carpeta "/var/www/html" (tot i que aquesta ubicació seria preferible que fos diferent però per simplicitat farem que sigui aquesta) amb el següent contingut. ¿Quina creus que pot ser la seva utilitat?

```
CREATE DATABASE IF NOT EXISTS exercici1;
USE exercici1;
CREATE TABLE IF NOT EXISTS Usuaris (
   Id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   Nom VARCHAR(30) NOT NULL,
   Cognom VARCHAR(30) NOT NULL,
   Email VARCHAR(50) NOT NULL,
   Data TIMESTAMP
);
```

> NOTA: La clausula "IF NOT EXISTS" a *CREATE DATABASE* s'ha especificat per a què si es torna a executar el mateix codi "init.sql" després d'un primer cop (és a dir, després d'haver-se creat ja la base de dades) no es dispari cap error (cosa que passaria si la cláusula no hi fos). Comentar, d'altra banda, que també existeix la possiblitat d'escriure CREATE ***OR REPLACE*** DATABASE ..., el qual seria equivalent a fer *DROP DATABASE IF EXISTS nomBD; CREATE DATABASE nomBD ...*
> NOTA: La clausula "IF NOT EXISTS" a *CREATE TABLE* s'ha especificat per a què si es torna a executar el mateix codi "init.sql" després d'un primer cop (és a dir, després d'haver-se creat ja la taula) no es dispari cap error (cosa que passaria si la cláusula no hi fos). Comentar, d'altra banda, que també existeix la possiblitat d'escriure CREATE ***OR REPLACE*** *TABLE* ..., el qual seria equivalent a fer *DROP TABLE IF EXISTS nomTaula; CREATE TABLE nomTaula ...*

b) Crea-hi un arxiu anomenat "config.php" dins de la carpeta "/var/www/html" amb el següent contingut (substitueix les "XXX" per la contrasenya que tinguis assignada a l'usuari "root" del servidor MariaDB) ¿Quina penses que pot ser la seva utilitat?

```
<?php
$host        = "localhost";
$username  = "root";
$password  = "XXX";
$dbname    = "exercici1";
$dsn         = "mysql:host=$host;dbname=$dbname";
$options    = array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION);
?>
```

c) Crea-hi un arxiu anomenat "install.php" dins de la carpeta "/var/www/html" amb el següent contingut ¿Quina penses que pot ser la seva utilitat?

```
<?php
require("config.php");
try {
  $conn = new PDO("mysql:host=$host", $username, $password, $options);
  $sql = file_get_contents("init.sql");
  $conn->exec($sql);
  echo "La base de dades i la taula s'han creat correctament.";
} catch(PDOException $e) {
  echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
}
$conn = null;
?>
```

d) Obre un navegador i vés a la URL http://ip.serv.Apache/install.php . ¿Què passa?

e) Crea-hi un arxiu anomenat "comprovacioInicial.php" dins de la carpeta "/var/www/html" amb el següent contingut i accedeix a ell a través del navegador. ¿Què veus, i per què?

```
<?php
function retrieveAndRenderResult(PDO $conn, string $sql) : void {
  $res=$conn->query($sql);
  echo "<table>";
  while ($fila=$res->fetch(PDO::FETCH_ASSOC)){
     echo "<tr>";
     foreach ($fila as $clau => $valor){ echo "<td>" . $valor . "</td>";  }
     echo "</tr>";
  }
  echo "</table><hr>";
}
require("config.php");
try {
  $conn = new PDO("mysql:host=$host", $username, $password, $options);
  retrieveAndRenderResult($conn,"SHOW DATABASES;");
  $conn->exec("USE exercici1;");
  retrieveAndRenderResult($conn,"SHOW TABLES;");
  retrieveAndRenderResult($conn,"DESCRIBE Usuaris;");
} catch(PDOException $e) {
  echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
}
$conn = null;
?>
```

2.-a) Crea un arxiu anomenat "header.html" dins de la carpeta "/var/www/html" amb el següent contingut....

```
<!DOCTYPE html><html><body>
<h1>Exemple d'interacció amb BBDDs mitjançant PDO</h1>
```

... i un altre anomenat "footer.html" amb aquest altre contingut:

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

b) Crea un arxiu anomenat "index.php" dins de la carpeta "/var/www/html" amb el següent contingut. Tot seguit, obre un navegador i vés a la URL http://ip.serv.Apache/index.php . ¿Què veus i per què?

```
<?php include("header.html"); ?>
<ul>
<li><a href="create.php">Crear un usuari</a></li>
<li><a href="search.php">Buscar un usuari</a></li>
</ul>
<?php include("footer.html"); ?>
```

c) Crea un arxiu anomenat "create.php" dins de la carpeta "/var/www/html" amb el següent contingut. ¿Quina penses que pot ser la seva utilitat (tant el codi PHP con el codi HTML, ja que és un "dos per un")?

```
<?php
include("header.html");
/*Aquest if comprova si hem arribat a la pàgina simplement navegant-hi o bé després de pulsar el botó d'enviar.
En el primer cas, es mostrarà només el formulari HTML; en el segon cas, es farà el processament de les dades
rebudes amb la base de dades (i tot seguit es tornarà a mostrar igual el formulari HTML) */
if (isset($_POST["submitcreate"])) {
  //No cal fer sanitització dels valors rebuts en l'array $_POST perquè farem servir "prepared statements"
  $nu = array("knom" => $_POST["firstname"], "kcog" => $_POST["lastname"],"kemail" => $_POST["email"]);
```

```
  $sql="INSERT INTO Usuaris (nom,cognom,email) VALUES (:knom,:kcog,:kemail);";
  try {
      require("config.php");
      $conn = new PDO($dsn, $username, $password, $options);
      $stmt = $conn->prepare($sql);
      $stmt->execute($nu);
      echo ("Creació d'usuari exitosa");
  } catch(PDOException $e) {
      echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
  }
  $conn=null;
}
?>
<h2>Crea un usuari</h2>
<form method="post" action="">
      Nom: <input type="text" name="firstname"> <br>
      Cognom: <input type="text" name="lastname"><br>
      Adreça electrònica: <input type="email" name="email"><br>
      <button type="submit" name="submitcreate">Crear</button>
</form>
<a href="index.php">Torna a l'inici</a>
<?php include("footer.html "); ?>
```

NOTA: Al codi anterior li faltaria fer una comprovació prèvia abans d'executar l'INSERT, que és la de veure si ja n'hi ha guardat a la base de dades un registre clon a l'actual (o si més no, amb algun camp amb un valor idèntic i que no volguem que ho sigui, com ara el email -la clau primària, en ser autonumèrica mai coincidirà-). Per fer això caldria fer una consulta per veure si s'obté (o no) algun registre clon al que es pretén insertar. Es deixa com exercici (en aquest sentit, es pot estudiar l'ús de la clàusula REPLACE https://mariadb.com/kb/en/replace/ o també INSERT ON DUPLICATE KEY UPDATE, https://mariadb.com/kb/en/insert-on-duplicate-key-update i IGNORE https://mariadb.com/kb/en/ignore/)

d) Crea un arxiu anomenat "search.php" dins de la carpeta "/var/www/html" amb el següent contingut. ¿Quina penses que pot ser la seva utilitat (tant el codi PHP con el codi HTML, ja que és un "dos per un")?

```
<?php
include("header.html");
if (isset($_POST["submitsearch"])) {
  $cogn = $_POST["lastname"];
  $sql="SELECT * FROM Usuaris WHERE Cognom = :kcog;" ;
  try {
    require("config.php");
    $conn = new PDO($dsn, $username, $password, $options);
    $stmt = $conn->prepare($sql);
    $stmt->bindParam(":kcog", $cogn, PDO::PARAM_STR);
    $stmt->execute();
    if ($stmt->rowCount() > 0) {
          echo("<h2>Resultats</h2>");
          echo("<table><thead><tr><th>Nom</th><th>Cognom</th><th>Adreça</th></tr></thead><tbody>");
          while ($fila=$stmt->fetch(PDO::FETCH_ASSOC)) {
              echo("<tr><td>".$fila["Nom"]."</td><td>".$fila["Cognom"]."</td><td>".$fila["Email"]." </td></tr>");
          }
          echo("</tbody></table>");
    } else  {  echo("No s'han trobat resultats");   }
  } catch(PDOException $e) {
    echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
  }
$conn=null;
}
?>
```

```
<h2>Buscar un usuari</h2>
<form method="post">
      Cognom a buscar: <input type="text" name="lastname">
      <button type="submit" name="submitsearch">Veure resultats</button>
</form>
<a href="index.php">Torna a l'inici</a>
<?php include("footer.html"); ?>
```

e) Obre un navegador i vés a la URL http://ip.serv.Apache/index.php i, a partir d'allà, crea tres usuaris, un anomenat "Pere Perez", un altre "Anton Lopez" i un altre "Estela Gomez". Un cop creats, fes una recerca per cadascun dels seus cognoms a veure si hi apareixen al buscador.

3.-a) Afegeix ara un nou enllaç a la llista d'enllaços mostrada a la pàgina "index.php" que tenim creada de l'exercici anterior (concretament un que apuntia una nova pàgina que anomenarem "list-to-update.php"). És a dir, fer que ara el codi de "index.php" sigui el següent (les novetats estan indicades en negreta):

```
<?php include("header.html"); ?>
<ul>
<li><a href="create.php">Crear un usuari</a></li>
<li><a href="search.php">Buscar un usuari</a></li>
<li><a href="list-to-update.php">Editar un usuari</a></li>
</ul>
<?php include("footer.html"); ?>
```

b) Crea dins de la carpeta "/var/www/html" l'arxiu anomenat "list-to-update.php" amb el següent contingut (la idea és que mostri tots els usuaris en una taula, acompanyant cadascun d'ells amb un enllaç anomenat "Edita" que apunti a una nova pàgina que permetrà modificar les característiques de l'usuari en qüestió de forma individual). Confirma, tot llegint el següent codi, que entens com està implementada aquesta funcionalitat i concretament, les parts marcades en negreta (ja que, en realitat, el codi és molt semblant al ja vist a l'exercici anterior corresponent a l'arxiu "search.php" i només s'han marcat en negreta els canvis més rellevants):

```
<?php
include("header.html");
$sql="SELECT * FROM Usuaris;" ;
try {
    require("config.php");
    $conn = new PDO($dsn, $username, $password, $options);
    $stmt = $conn->prepare($sql);
    $stmt->execute();
    if ($stmt->rowCount() > 0) {
       echo("<h2>Llista d'usuaris</h2>");
       echo("<table><thead><tr><th>N</th><th>C</th><th>A</th><th>Edita</th></tr></thead><tbody>");
       while ($fila=$stmt->fetch(PDO::FETCH_ASSOC)) {
            echo("<tr><td>".$fila["Nom"]."</td><td>".$fila["Cognom"]."</td><td>".$fila["Email"]."</td>
<td><a href='update-single.php?id=".$fila["Id"]."'>Edita</a></td></tr>");
       }
       echo("</tbody></table>");
    } else  {  echo("No s'han trobat resultats");   }
} catch(PDOException $e) {
    echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
}
$conn=null;
?>
<a href="index.php">Torna a l'inici</a>
<?php include("footer.html"); ?>
```

c) Crea dins de la carpeta "/var/www/html" un arxiu anomenat "update-single.php" amb el següent contingut A partir del seu codi (similar al de la pàgina "search.php") i del codi mostrat a l'apartat anterior, ¿quina penses que pot ser la seva utilitat?

```
<?php
include("header.html");
/*Aquest if comprova si hem arribat a la pàgina simplement navegant-hi o bé després de pulsar el link d'editar de la pàgina "update.php". En el primer cas, es mostrarà el formulari HTML amb les dades actuals per poder-les modificar i actualitzar-les; en el segon cas, es mostrarà un simple missatge d'error */
if (isset($_GET["id"])) {
  $id = $_GET["id"];
  $sql="SELECT * FROM Usuaris WHERE Id = :id;";
  try {
      require("config.php");
      $conn = new PDO($dsn, $username, $password, $options);
      $stmt = $conn->prepare($sql);
      $stmt->bindParam(":id", $id, PDO::PARAM_INT);
      $stmt->execute();
      $user = $stmt->fetch(PDO::FETCH_ASSOC);
      echo("<h2>Modifica un usuari</h2>");
      echo("<form method='post' action='update-single-do.php?id=".$id."'>");
      echo("Nom: <input type='text' name='firstname'  value='". $user["Nom"]."'><br>");
      echo("Cognom: <input type='text' name='lastname' value='".$user["Cognom"]."'><br>");
      echo("Adreça electrònica: <input type='email' name='email' value='".$user["Email"]."'><br>");
      echo("<button type='submit' name='submitmodify'>Modificar</button>");
      echo("</form>");
  } catch(PDOException $e) {
      echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
  }
  $conn=null;
}
?>
<a href="index.php">Torna a l'inici</a>
<?php include("footer.html"); ?>
```

NOTA: Pensa com podries afegir, en aquesta pàgina, la possibilitat de cancelar una eventual modificació a mig fer

d) Crea dins de la carpeta "/var/www/html" un arxiu anomenat "update-single-do.php" amb el següent contingut (que serà molt semblant a "create.php"...les parts més significatives que difereixen s'han marcat en negreta). ¿Quina penses que pot ser la seva utilitat?

```
<?php
include("header.html");
if (isset($_POST["submitmodify"])) {
  $mu = array("kid" => $_GET["id"], "knom" => $_POST["firstname"], "kcog" => $_POST["lastname"],"kemail" => $_POST["email"]);
  $sql="UPDATE Usuaris SET Nom = :knom , Cognom = :kcog , Email = :kemail WHERE Id = :kid;";
  try {
      require("config.php");
      $conn = new PDO($dsn, $username, $password, $options);
      $stmt = $conn->prepare($sql);
      $stmt->execute($mu);
      echo ("Modificació d'usuari exitosa <br>");
  } catch(PDOException $e) {
      echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
  }
  $conn=null;
}
?>
<a href="index.php">Torna a l'inici</a>
<?php include("footer.html"); ?>
```

e) Obre un navegador i vés a la URL http://ip.serv.Apache/index.php i, a partir d'allà, modifica el nom i/o el cognom i/o l'email d'algun dels usuaris ja existents. Un cop fetes aquests canvis, comprova que s'hi han guardat visualitzant de nou les seves dades.

eII) Accedeix al SGBD MariaDB a través de la comanda client *mariadb* per comprovar el valor de la columna de tipus TIMESTAMP que existeix a la taula "Usuaris". ¿Quins són els valors que conté i per què?

4.-a) Afegeix ara un nou enllaç a la llista d'enllaços mostrada a la pàgina "index.php" que ja teníem creada d'exercicis anteriors (concretament un que apuntia una nova pàgina que anomenarem "delete.php"). És a dir, fer que ara el codi de "index.php" sigui el següent (les novetats estan indicades en negreta):

```
<?php include("header.html"); ?>
<ul>
<li><a href="create.php">Crear un usuari</a></li>
<li><a href="search.php">Buscar un usuari</a></li>
<li><a href="list-to-update.php">Editar un usuari</a></li>
<li><a href="list-to-delete.php">Eliminar un usuari</a></li>
</ul>
<?php include("footer.html"); ?>
```

b) Copia tot el contingut de l'arxiu "list-to-update.php" a un altre arxiu, també dins de la carpeta "/var/www/html", anomenat "list-to-delete.php" i canvia el títol de la columna de més a la dreta per a què sigui "Elimina" (en lloc de "Edita"), així com el nom de la pàgina on els enllaços corresponents apunten per a què sigui "delete-single-do.php" (en lloc de "update-single.,php") i el text de l'enllaç per a què sigui "Elimina" (en lloc d'"Edita"). Tota la resta ho pots deixar igual.

c) Crea dins de la carpeta "/var/www/html" un arxiu anomenat "delete-single-do.php" amb el següent contingut (que serà molt semblant a "update-single-do.php"...les parts més significatives que difereixen s'han marcat en negreta). ¿Quina penses que pot ser la seva utilitat?

```
<?php
include("header.html");
if (isset($_GET["id"])) {
  $id = $_GET["id"];
  $sql="DELETE FROM Usuaris WHERE Id = :id;";
  try {
      require("config.php");
      $conn = new PDO($dsn, $username, $password, $options);
      $stmt = $conn->prepare($sql);
      $stmt->bindParam(":id", $id, PDO::PARAM_INT);
      $stmt->execute();
      echo ("Eliminació d'usuari exitosa <br>");
  } catch(PDOException $e) {
      echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
  }
  $conn=null;
}
?>
<a href="index.php">Torna a l'inici</a>
<?php include("footer.html"); ?>
```

d) Obre un navegador i vés a la URL http://ip.serv.Apache/index.php i, a partir d'allà, elimina algun dels usuaris ja existents. Comprova que s'han eliminat realment visualitzant de nou la llista d'usuaris existents.

e) Canvia el codi de la pàgina "delete-single-do.php" per a què ara sigui aquest (s'ha marcat en negreta els canvis més importants)¿Què passa ara quan elimines un usuari des de la pàgina "list-to-delete.php" i per què?

```
<?php
if (isset($_GET["id"])) {
  $id = $_GET["id"];
  $sql="DELETE FROM Usuaris WHERE Id = :id;";
  try {
      require("config.php");
      $conn = new PDO($dsn, $username, $password, $options);
      $stmt = $conn->prepare($sql);
      $stmt->bindParam(":id", $id, PDO::PARAM_INT);
      $stmt->execute();
  } catch(PDOException $e) {
      echo "Ha ocorregut un error en contactar amb la BD: " . $e->getMessage();
  }
  $conn=null;
}
header("Location:list-to-delete.php");
?>
```

NOTA: Fixa't com abans de cridar a la funció header() no s'envia cap caràcter a la sortida estàndard (via *echo()* o similar); això és un requisit imprescindible per a què la redirecció automàtica funcioni.

5.-Modifica el lloc web desenvolupat al llarg dels exercicis anteriors per tal que ara la pàgina "index.php" mostri els següents elements:

* Una taula amb quatre columnes: les dues primeres mostraran respectivament el nom i els cognoms de tots els usuaris existents a la base de dades, la tercera mostrarà l'enllaç "Edita" i la quarta mostrarà l'enllaç "Elimina". En el cas de pulsar sobre l'enllaç "Edita", caldrà que aparegui una nova pàgina amb un formulari que permeti modificar el nom, cognoms o adreça de l'usuari en qüestió, retornant a la pàgina "index.php" quan es confirmin els canvis. En el cas de pulsar sobre l'enllaç "Elimina", caldrà actualitzar directament el contingut mostrat a la pàgina "index.php"

* Un botó que permeti crear un usuari nou (mostrant un formulari pertinent i retornant a la pàgina "index.php" un cop confirmada l'addició)

* Una caixa de text (acompanyada d'un botó) que permeti filtrar (un cop pulsat el botó) els usuaris mostrats a la taula segons el valor que s'hi hagi escrit a la caixa, valor que ha de coincidir amb el d'algun dels cognoms dels usuaris existents.

6.-Modifica el lloc web desenvolupat a l'exercici anterior per tal que ara:

* La pàgina "index.php" sigui un formulari "de login" que demani una adreça electrònica (amb una caixa de tipus "email" i un botó d'enviar). Aquest formulari comprovarà que l'adreça introduïda es correspongui a la d'algun usuari existent en la base de dades. Si no fos així, hauria de tornar-se a veure el mateix formulari de login juntament amb un missatge d'error del tipus "No s'ha trobat l'adreça introduïda" o similar. Si sí es trobés, es mostrarà llavors la pàgina "index2.php"

* La pàgina "index2.php" serà el que era l'antiga pàgina "index.php" a l'exercici anterior. No obstant, els enllaços "Edita" i "Elimina" no hauran de ser visibles a no ser que l'adreça electrònica introduïda en el formulari de login hagi sigut "admin@admin.com" (en aquest cas, sí que es seran operatius). En tot cas, ha de mostrar sempre un enllaç "Tancar sessió", que retornarà a "index.php". Fes el necessari per a què aquesta pàgina no pugui ser visible si no l'usuari no s'ha loguejat prèviament.

PISTA: Fes servir variables de sessió per controlar en tot moment si l'usuari loguejat és "admin" o no