Web technologies

PHP Data Objects (PDO)

David Jelenc

Index

Index

Database Management Systems

Advantages over saving data directly to files:

Database Management Systems

Advantages over saving data directly to files:

- increases data independence

Database Management Systems

Advantages over saving data directly to files:

- increases data independence
- improves data access

Database Management Systems

Advantages over saving data directly to files:

- increases data independence
- improves data access
- improves security and data integrity

Database Management Systems

Advantages over saving data directly to files:

- increases data independence
- improves data access
- improves security and data integrity
- eases administration, and thus reduces the development time

Database Management Systems

Advantages over saving data directly to files:

- increases data independence
- improves data access
- improves security and data integrity
- eases administration, and thus reduces the development time

In this class

Database Management Systems

Advantages over saving data directly to files:

- increases data independence
- improves data access
- improves security and data integrity
- eases administration, and thus reduces the development time

In this class

- We will be using MariaDB (10.1.21)
  - Part of the XAMPP installation
  - Fork of MySQL
  - `https://mariadb.org`

Database Management Systems

Advantages over saving data directly to files:

- increases data independence
- improves data access
- improves security and data integrity
- eases administration, and thus reduces the development time

In this class

- We will be using MariaDB (10.1.21)
  - Part of the XAMPP installation
  - Fork of MySQL
  - `https://mariadb.org`
- Mostly compliant with ANSI SQL standard
  `https://mariadb.com/kb/en/mariadb/sql-mode`

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**
- Usually, a MariaDB installation consists of multiple databases of which each contains a number of related tables

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**
- Usually, a MariaDB installation consists of multiple databases of which each contains a number of related tables
  - E.g. database `web_store` contains tables like `customer`, `item`, `invoice` and others

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**
- Usually, a MariaDB installation consists of multiple databases of which each contains a number of related tables
  - E.g. database `web_store` contains tables like `customer`, `item`, `invoice` and others
  - There are multiple meta databases (`information_schema`, `root`) that contain metadata about the database: names of tables, access rules etc.

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**
- Usually, a MariaDB installation consists of multiple databases of which each contains a number of related tables
  - E.g. database `web_store` contains tables like `customer`, `item`, `invoice` and others
  - There are multiple meta databases (`information_schema`, `root`) that contain metadata about the database: names of tables, access rules etc.
- **Access control** is based on `username`/`password`:

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**
- Usually, a MariaDB installation consists of multiple databases of which each contains a number of related tables
  - E.g. database `web_store` contains tables like `customer`, `item`, `invoice` and others
  - There are multiple meta databases (`information_schema`, `root`) that contain metadata about the database: names of tables, access rules etc.
- **Access control** is based on `username`/`password`:
  - Special user `root` can modify all aspects of a general RDMS

Database structure

- Every installation of MariaDB contains multiple **databases** (or schemes)
- Every scheme typically contains multiple **tables**
- Usually, a MariaDB installation consists of multiple databases of which each contains a number of related tables
  - E.g. database `web_store` contains tables like `customer`, `item`, `invoice` and others
  - There are multiple meta databases (`information_schema`, `root`) that contain metadata about the database: names of tables, access rules etc.
- **Access control** is based on `username/password`:
  - Special user `root` can modify all aspects of a general RDMS
  - New users ca be added with SQL commands or with dedicated administration tools

Database structure

Data can be stored in various ways, depending on the **storage engine**:

Database structure

Data can be stored in various ways, depending on the **storage engine**:

- **XtraDB, InnoDB**: supports transactions (ACID), stored procedures, triggers, foreign key constraints

Database structure

Data can be stored in various ways, depending on the **storage engine**:

- **XtraDB, InnoDB**: supports transactions (ACID), stored procedures, triggers, foreign key constraints
- **Aria, MyISAM**: small footprint, allows for easy copying between systems (default in the past)

Database structure

Data can be stored in various ways, depending on the **storage engine**:

- **XtraDB, InnoDB**: supports transactions (ACID), stored procedures, triggers, foreign key constraints
- **Aria, MyISAM**: small footprint, allows for easy copying between systems (default in the past)
- Many others: Memory, Merge, Archive, Federated, NDBCLUSTER, CSV, Blackhole, Example

Database structure

Data can be stored in various ways, depending on the **storage engine**:

- **XtraDB, InnoDB**: supports transactions (ACID), stored procedures, triggers, foreign key constraints
- **Aria, MyISAM**: small footprint, allows for easy copying between systems (default in the past)
- Many others: Memory, Merge, Archive, Federated, NDBCLUSTER, CSV, Blackhole, Example
- Storage engine can be determined on the table level

Connecting to the database

- The DBMS is listening on a TCP socket on port 3306

Connecting to the database

- The DBMS is listening on a TCP socket on port 3306
- To connect we use:

Connecting to the database

- The DBMS is listening on a TCP socket on port 3306
- To connect we use:
  - a terminal and write `$ mysql -u <username> -p`

Connecting to the database

- The DBMS is listening on a TCP socket on port 3306
- To connect we use:
  - a terminal and write `$ mysql -u <username> -p`
  - a dedicated tool like MySQL Workbench or phpMyAdmin

An SQL query example

- To create a database `web_store`, a user `student` whose password is `password123` (valid only when connecting from `localhost`) and grant her proper credentials to work with the database

An SQL query example

- To create a database `web_store`, a user `student` whose password is `password123` (valid only when connecting from `localhost`) and grant her proper credentials to work with the database

```sql
CREATE DATABASE web_store;
CREATE USER 'student'@'localhost' IDENTIFIED BY '
    ↪ password123';
GRANT ALL ON web_store.* TO 'student'@'localhost';
```

Listing 2: SQL - Creating user and granting access

Index

Using function that start with mysql_

- The **old way** of accessing a MySQL (or MariaDB) database was by using functions that start with `mysql_`:

Using function that start with mysql_

- The **old way** of accessing a MySQL (or MariaDB) database was by using functions that start with `mysql_`:
  - `mysql_connect()`
  - `mysql_select_db()`
  - `mysql_query()`
  - `mysql_close()`

Using function that start with mysql_

- The **old way** of accessing a MySQL (or MariaDB) database was by using functions that start with `mysql_`:
  - `mysql_connect()`
  - `mysql_select_db()`
  - `mysql_query()`
  - `mysql_close()`
- **Do not use them.** They are obsolete, because:

Using function that start with mysql_

- The **old way** of accessing a MySQL (or MariaDB) database was by using functions that start with `mysql_`:
  - `mysql_connect()`
  - `mysql_select_db()`
  - `mysql_query()`
  - `mysql_close()`
- **Do not use them.** They are obsolete, because:
  - they are specific to MySQL (and MariaDB) and do not support other RDBMS,

Using function that start with mysql_

- The **old way** of accessing a MySQL (or MariaDB) database was by using functions that start with `mysql_`:
  - `mysql_connect()`
  - `mysql_select_db()`
  - `mysql_query()`
  - `mysql_close()`
- **Do not use them.** They are obsolete, because:
  - they are specific to MySQL (and MariaDB) and do not support other RDBMS,
  - they are considered less secure (SQL injection).

Using mysql_connect(), mysql_select_db()

```php
$dbcnx = mysql_connect("localhost", "root", "");
if (!$dbcnx) {
  die("Connection failed.");
}

if (!@mysql_select_db("jokes", $dbcnx)) {
  die("There is no such table as jokes.");
} else {
  echo "Selected table jokes.";
}
```

Listing 3: PHP - Connecting and selecting a table

Using mysql_query(), mysql_fetch_array()

```php
$result = @mysql_query("SELECT * FROM jokes", $dbcnx);

if (!$result) {
  die('Query failed: ' . mysql_error());
}

while ($row = mysql_fetch_array($result))
  echo "$row[id]: $row[joke_text]\n";

$sql = "INSERT INTO jokes SET joke_text='There are only
    ↪ 10 types of people in the world. Those who
    ↪ understand binary and those who do not', joke_date=
    ↪ CURDATE()";

if (@mysql_query($sql, $dbcnx))
  echo("Joke added.");
```

Listing 4: PHP - Reading and inserting

PHP Data Objects (PDO)

- A library that provides a **unified access** to multiple RDBMS

PHP Data Objects (PDO)

- A library that provides a **unified access** to multiple RDBMS:
  - PDO_CUBRID PDO_DBLIB, PDO_FIREBIRD, PDO_IBM, PDO_INFORMIX, PDO_MYSQL, PDO_OCI, PDO_ODBC, PDO_PGSQL, PDO_SQLITE, PDO_SQLSRV, PDO_4D

PHP Data Objects (PDO)

- A library that provides a **unified access** to multiple RDBMS:
  - PDO_CUBRID PDO_DBLIB, PDO_FIREBIRD, PDO_IBM, PDO_INFORMIX, PDO_MYSQL, PDO_OCI, PDO_ODBC, PDO_PGSQL, PDO_SQLITE, PDO_SQLSRV, PDO_4D
- Afterwards, the programmer can **freely migrate** between these RDBMS

PHP Data Objects (PDO)

- A library that provides a **unified access** to multiple RDBMS:
  - PDO_CUBRID PDO_DBLIB, PDO_FIREBIRD, PDO_IBM, PDO_INFORMIX, PDO_MYSQL, PDO_OCI, PDO_ODBC, PDO_PGSQL, PDO_SQLITE, PDO_SQLSRV, PDO_4D
- Afterwards, the programmer can **freely migrate** between these RDBMS
- PDO offers an object-oriented interface

PHP Data Objects (PDO)

- A library that provides a **unified access** to multiple RDBMS:
  - PDO_CUBRID PDO_DBLIB, PDO_FIREBIRD, PDO_IBM, PDO_INFORMIX, PDO_MYSQL, PDO_OCI, PDO_ODBC, PDO_PGSQL, PDO_SQLITE, PDO_SQLSRV, PDO_4D
- Afterwards, the programmer can **freely migrate** between these RDBMS
- PDO offers an object-oriented interface
- PDO is considered more secure than functions `mysql_`

PHP Data Objects (PDO)

- A library that provides a **unified access** to multiple RDBMS:
  - PDO_CUBRID PDO_DBLIB, PDO_FIREBIRD, PDO_IBM, PDO_INFORMIX, PDO_MYSQL, PDO_OCI, PDO_ODBC, PDO_PGSQL, PDO_SQLITE, PDO_SQLSRV, PDO_4D
- Afterwards, the programmer can **freely migrate** between these RDBMS
- PDO offers an object-oriented interface
- PDO is considered more secure than functions `mysql_`

PDO: Connecting to the database

```php
try {
  // connecting
  $dbh = new PDO("mysql:host=localhost;dbname=database", "username", "password");
  // setting error reporting
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch(PDOException $e) {
  echo "Failed to connect: {$e->getMessage()}";
}

// disconnecting
$dbh = null;
```

Listing 5: PHP - Connecting to the database

PDO: Issuing SQL queries

```php
try {
  $stmt = $dbh->query("SELECT id, joke_text FROM jokes");

  foreach ($stmt as $row) {
    echo "$row[id]: $row[joke_text]\n";
  }
} catch (PDOException $e) {
  echo "An error occurred: {$e->getMessage()}";
}
```

Listing 6: PHP - Issuing SELECT queries

PDO: Querying with positional parameters

```php
try {
  $stmt = $dbh->prepare("SELECT id, joke_text FROM jokes
      ↪ WHERE joke_text LIKE ?");
  $stmt->bindValue(1, "%chuck%");
  $stmt->execute();

  foreach ($stmt->fetchAll() as $row) {
    echo "$row[id]: $row[joke_text]\n";
  }
} catch (PDOException $e) {
  echo "An error occurred: {$e->getMessage()}";
}
```

Listing 7: PHP - Querying with positional parameters

PDO: Inserting

```php
try {
  $stmt = $dbh->prepare("INSERT INTO jokes (joke_text, joke_date) VALUES (?, ?)");
  $stmt->bindValue(1, "Chuck Norris can write infinite recursion functions ... and have them return.");
  $stmt->bindValue(2, "2017-04-10");
  $stmt->execute();

  echo "Joke added, id = {$dbh->lastInsertId()}";
} catch (PDOException $e) {
  echo "An error occurred: {$e->getMessage()}";
}
```

Listing 8: PHP - Inserting

PDO: Querying with named parameters

```php
try {
    $stmt = $dbh->prepare("DELETE FROM jokes WHERE id = :id ");
    $stmt->bindValue(":id", 1, PDO::PARAM_INT);
    $stmt->execute();

    echo "Number of affected rows: {$stmt->rowCount()}.";
} catch (PDOException $e) {
    echo "An error occurred: {$e->getMessage()}.";
}
```

Listing 9: PHP - Deleting with named parameters

PDO: Updating and using bindParam()

```php
try {
  $stmt = $dbh->prepare("UPDATE jokes SET joke_text = :joke_text, joke_date = :joke_date WHERE id = :id");
  $stmt->bindParam(":joke_text", $text);
  $stmt->bindParam(":joke_date", $date);
  $stmt->bindValue(":id", 3, PDO::PARAM_INT);

  $text = "All arrays Chuck Norris declares are of infinite size, because Chuck Norris knows no bounds.";
  $date = "2017-04-10";

  $stmt->execute();
} catch (PDOException $e) {
  echo "An error occurred: {$e->getMessage()}";
}
```

Listing 10: PHP - Updating and using bindParam()

Index

XAMPP and MariaDB

- RDBMS MariaDB is part of the core **XAMPP** package
- Additionally, you also get a PHP application called **phpMyAdmin** for working with the database
  - `http://www.phpmyadmin.net`
- The application should be accessible on `http://localhost/phpmyadmin`
- By default, you can log-in with user `root` without providing password