Importing CSV into MySQL with PHP

Build a secure administrator-only workflow that validates fictional CSV data and stores acceptable rows in MySQL.

Understanding CSV Data

CSV means comma-separated values. Each line is a record and each position represents a field.

```
match_id,team_name,wins,losses
M001,Fictional Falcons,4,1
M002,Sample Sharks,3,2
```

The first row is a header. Quoted fields may contain commas, so do not process CSV with `explode(",")`; use `fgetcsv()`.

Inspect before importing

Identify headers, data types, required fields, allowed ranges, unique identifiers, missing values and any fields that relate to other tables.

Treat every upload as untrusted input. A file being supplied with a task does not remove the need for validation.

Check

- [ ] Headers have meaning and match expected fields.
- [ ] Every row has the expected number of columns.
- [ ] Numeric and identifier rules are documented.
- [ ] All classroom records are fictional.

Designing a MySQL Table from a CSV

Design the destination table from the meaning of the data, not merely its appearance in a spreadsheet.

CSV field	MySQL type	Rule
match_id	VARCHAR(20)	unique and required
team_name	VARCHAR(100)	required
wins	INT	zero or greater
losses	INT	zero or greater

```
CREATE TABLE team_results (
  result_id INT AUTO_INCREMENT PRIMARY KEY,
  match_id VARCHAR(20) NOT NULL UNIQUE,
  team_name VARCHAR(100) NOT NULL,
  wins INT NOT NULL,
  losses INT NOT NULL
);
```

A surrogate primary key supports database operations; the unique source identifier prevents accidental duplicates.

Design questions

- Does one CSV row represent one entity or a relationship?
- Which values repeat independently and belong in related tables?
- Which constraints protect data quality?
- Will repeated imports insert, reject or update existing records?

Document and justify the chosen policy.

Check

- [ ] Types and lengths fit expected values.
- [ ] Required fields are constrained.
- [ ] Duplicate behaviour is defined.
- [ ] Relationships reflect the solution’s data needs.

Creating an Admin-Only CSV Upload Form

A CSV upload changes stored data and should be restricted to administrators. Apply session and role checks before any output.

```
<?php
session_start();
if (!isset($_SESSION["user_id"])) {
    header("Location: login.php");
    exit;
}
if (($_SESSION["role"] ?? "") !== "admin") {
    http_response_code(403);
    exit("Permission denied.");
}
?>
<form method="post" enctype="multipart/form-data">
  <label for="dataset">CSV dataset</label>
  <input id="dataset" name="dataset" type="file" accept=".csv,text/csv" required>
  <button type="submit">Validate and import</button>
</form>
```

The `multipart/form-data` encoding is required. The accept attribute guides file selection but does not provide server-side security.

Check

- [ ] Logged-out and standard users are rejected.
- [ ] The form has a visible label.
- [ ] File input has a restrictive accept hint.
- [ ] Processing repeats the server-side role check.

Validating an Uploaded CSV File

Validate the upload before reading its rows.

```
$file = $_FILES["dataset"] ?? null;
$errors = [];

if (!$file || $file["error"] !== UPLOAD_ERR_OK) {
    $errors[] = "Choose a CSV file that uploaded successfully.";
}
if ($file && $file["size"] > 2 * 1024 * 1024) {
    $errors[] = "The file must be 2 MB or smaller.";
}
$extension = $file ? strtolower(pathinfo($file["name"], PATHINFO_EXTENSION)) : "";
if ($extension !== "csv") {
    $errors[] = "The file must use the .csv extension.";
}
```

A filename or MIME type alone can be misleading. Combine upload status, size, extension, readable content, header and row validation.

Never use the original filename as a server path. Process the temporary upload and store only what the application requires.

Check

Test missing files, oversized files, wrong extensions, empty files and malformed content. Every failure should produce a clear message and must not partially import data.

Reading CSV Rows with fgetcsv

Use PHP’s CSV parser so quoted commas and escaped values are handled correctly.

```
$handle = fopen($file["tmp_name"], "r");
if ($handle === false) {
    exit("The uploaded file could not be read.");
}

$headers = fgetcsv($handle);
$expected = ["match_id", "team_name", "wins", "losses"];

if ($headers !== $expected) {
    fclose($handle);
    exit("The CSV headings do not match the expected format.");
}

$rowNumber = 1;
while (($row = fgetcsv($handle)) !== false) {
    $rowNumber++;
    // Validate and store this row.
}
fclose($handle);
```

This loop demonstrates iteration. The header comparison prevents values being assigned to the wrong fields.

Check

- [ ] File open failure is handled.
- [ ] The header is read separately.
- [ ] Exact expected headings are checked.

- [ ] Row numbers are tracked for useful feedback.
- [ ] The handle is closed.

Validating CSV Rows

Validate every row before storing it.

```
if (count($row) !== 4) {
    $rowErrors[] = "Row $rowNumber has the wrong number of fields.";
    continue;
}

[$matchId, $teamName, $winsRaw, $lossesRaw] = array_map("trim", $row);

$wins = filter_var($winsRaw, FILTER_VALIDATE_INT);
$losses = filter_var($lossesRaw, FILTER_VALIDATE_INT);

if ($matchId === "" || $teamName === "") {
    $rowErrors[] = "Row $rowNumber has a missing required value.";
    continue;
}
if ($wins === false || $losses === false || $wins < 0 || $losses < 0) {
    $rowErrors[] = "Row $rowNumber has invalid results.";
    continue;
}
```

The loop is iteration; each validation decision is selection. Preserve row numbers so an administrator can correct the source.

Define whether one bad row rejects the whole file or only that row. For assessed work, justify the policy from integrity and user needs.

Check

Test blank, extra, missing, non-numeric, negative and duplicate values.

Inserting Imported Rows Safely

Prepare the insert once, then execute it for each valid row.

```
$insert = $pdo->prepare(
    "INSERT INTO team_results (match_id, team_name, wins, losses)
     VALUES (:match_id, :team_name, :wins, :losses)"
);

$insert->execute([
    "match_id" => $matchId,
    "team_name" => $teamName,
    "wins" => $wins,
    "losses" => $losses
]);
```

Use a transaction when the intended policy is all-or-nothing:

```
$pdo->beginTransaction();
try {
    // Read, validate and insert every row.
    $pdo->commit();
} catch (Throwable $error) {
    $pdo->rollBack();
    throw $error;
}
```

Transactions prevent a failed whole-file import from leaving a partial update. If valid rows may be retained while invalid rows are rejected, count both outcomes and report them clearly.

Check

- [ ] Prepared statements separate values from SQL.

- [ ] Duplicate behaviour is handled.
- [ ] Transaction policy matches the intended outcome.
- [ ] Raw database errors are not shown publicly.

Reporting CSV Import Results

An administrator needs evidence of what happened, not just “upload complete”.

Report useful totals

Show the escaped filename, rows read, inserted, updated, skipped and rejected, plus row-specific validation messages and whether a transaction committed or rolled back.

```
$summary = [
  "read" => $rowsRead,
  "inserted" => $rowsInserted,
  "rejected" => count($rowErrors)
];
```

Totals must reconcile: rows read equals all outcome categories combined. Never display passwords, server paths or raw SQL errors.

Accessible output

Use a clear heading and a list or table. Do not communicate success through colour alone. Make corrective messages specific: include the CSV row number and violated rule.

Check

- [ ] Totals match database records.
- [ ] Both success and failure cases are tested.
- [ ] Messages help the administrator correct the file.
- [ ] Sensitive technical details remain hidden.

Building a Complete Admin CSV Import

A complete import combines access control, upload checks, parsing, row validation, prepared statements and a result summary.

Processing sequence

1. Start the session and require the administrator role.
2. Accept a POST upload.
3. Validate upload status, size and extension.
4. Open the temporary file and verify headers.
5. Begin the chosen transaction policy.
6. Iterate through rows.
7. Validate fields and store valid data.
8. Commit or roll back.
9. Close the file and display a summary.

Pseudocode

```
BEGIN
  REQUIRE administrator
  INPUT CSV
  IF file is invalid THEN OUTPUT error; STOP
  READ and CHECK header
  FOR EACH row
    VALIDATE fields
    IF valid THEN STORE with prepared statement
    ELSE RECORD row error
    ENDIF
  ENDFOR
  OUTPUT summary
END
```

This provides evidence of input, selection, iteration, storage and output. Separate repeated checks into functions when it improves readability.

Check

Trace one valid and one invalid row through every step. Explain the duplicate and partial-import policy rather than leaving either to chance.

Testing an Admin CSV Import

Testing must cover permissions, validation, database effects and feedback.

Test	Condition	Expected result
Logged out	Direct import URL	Redirect to login
Standard role	Direct URL or POST	403; nothing imported
Valid CSV	Correct headings and rows	Expected records stored
Wrong heading	Changed column	File rejected
Invalid number	Text or negative result	Stated rejection policy
Duplicate ID	Existing identifier	Stated duplicate policy
Empty file	No usable rows	Clear rejection
Oversized file	Above limit	Rejected before parsing

Record actual result, pass/fail and refinement. Verify the database rather than trusting the on-screen message.

Use fictional accounts and data. Hide credentials and unrelated information in screenshots.

Completion checklist

- ☐ Direct access and processing are protected.
- ☐ Invalid data cannot silently enter storage.
- ☐ Counts reconcile with database rows.
- ☐ Error messages support correction.
- ☐ Retesting confirms refinements.