● Identify the Problem:
What could be the possible reasons the import fails or becomes very slow?

My Answer :

1. Synchronous Processing

The main issue seems to be because the CSV upload process is done synchronously. So, from uploading the file, reading each CSV row, and inserting into the database — all are done in order, one after another. This makes the process wait for each step, and that causes everything to be very slow.

2. Server Has a Patience Limit (max_execution_time)

By default, PHP has a maximum execution time, usually 30 seconds. Processing 100,000 rows and doing DB inserts will probably go beyond that and cause a timeout error.

3. Memory Limit

PHP also has a setting called memory_limit, which limits how much memory a PHP process can use. This can also affect the speed or success of the import process.

● Propose a Solution:
How would you design an efficient import process that can handle large files? What methods would you use?

My Answer :

1. Use Queue

Using queues helps to avoid the timeout problem caused by PHP’s max execution time. If the process runs synchronously, it might go over the limit. But with queues, the program runs asynchronously in the background.

2. Use Batching & Chunking

Before, we tried to insert 100,000 rows all at once. But by using batching and chunking, we can break the file into smaller parts (for example, 1000 rows per batch). Then we insert each batch using bulk insert. This saves memory and makes the process more efficient.

3. Use Transaction

Each batch is processed inside a transaction. So, if there's an error during insert, we can roll back that batch only — not the whole process. This makes the import safer, because only the failed batch is affected, not everything.

● Plan the Implementation:
How would you validate the data and ensure consistency without blocking the main application?

My Answer :

Step 1: Separate Upload and Import Process

This helps the upload finish quickly because we’re not running parsing/inserting at the same time.

Step 2: Queue Job

After the file is uploaded, we immediately dispatch a queue job. This moves the heavy processing to the background so that users using the system aren't affected. Also, the job will run asynchronously, and we can have multiple workers processing in parallel.

Step 3: Initial Validation

Before running the big insert/parsing process, we should do a light validation — like checking if the CSV headers match the expected format, and making sure required data isn’t missing.

Step 4: Batch Processing

At this stage, we read the file per batch (like 1000 rows). This makes the process lighter and faster.

Step 5: Use Database Transaction

Here, each batch will be wrapped in a transaction block. So if an error happens in a batch, we can roll it back — but only that batch. Not the entire import.

● Prepare for Growth:
How would you handle importing millions of records in the future, while keeping the system stable?

My Answer :

1. Use Database Indexing

Using indexing can help speed up lookup during validation, and it can reduce latency when doing large inserts.

2. Use Efficient Tools for Big CSV Files

There are some tools that are made to handle large CSV imports. I think these can be very helpful and efficient to use for improving this feature in the future.