Concurrency & Locking in Laravel Databases

Why Concurrency Matters

When multiple requests hit the same records at the same time, race conditions can corrupt data. Concurrency control is how we keep totals correct, avoid double-charging, and prevent lost updates.

Typical Problems

- Lost updates: two users update the same row and one update overwrites the other.
- Dirty reads: a transaction reads uncommitted changes.
- Inconsistent reads: the same query returns different results inside a single flow.

Laravel Tools for Concurrency

1) Database Transactions

Use transactions to make a set of operations atomic.

```
DB::transaction(function () {
    // read
    // write
});
```

2) Pessimistic Locking (For Update)

Lock rows while you update them.

```
DB::transaction(function () {
    $wallet = DB::table('wallets')
        ->where('id', $id)
        ->lockForUpdate()
        ->first();

    // update safely
});
```

3) Shared Locks

Allow reads but block writes.

```
DB::transaction(function () {
    $report = DB::table('reports')
        ->where('id', $id)
        ->sharedLock()
        ->first();
});
```

4) Optimistic Locking (Manual)

Add a `version` column and check it before update.

```
$updated = DB::table('orders')
    ->where('id', $id)
    ->where('version', $version)
    ->update([
        'status' => 'paid',
        'version' => $version + 1,
    ]);

if ($updated === 0) {
    // someone else changed it, retry
}
```

Practical Tips

- Always wrap money updates in transactions.
- Use `lockForUpdate()` when you must guarantee a single writer.
- Keep transactions short to avoid long locks.

Summary

Concurrency is not optional in production. Laravel gives you transactions and row-level locking. Use them to protect integrity and reliability.