Group Practicum Jobsheet: Meeting 4

Database Design, Migrations, and Seeding (Group Work)

Course	Advanced Web Programming (SIB245007)
Meeting	4 (Week 4)
Duration	2 sessions × 170 minutes
Sub-CPMK	Sub-CPMK 1: Students can understand the basic concepts of web frameworks and apply routing, controllers, and database management in web application development.
Work Mode	Group (same as Meeting 3), one shared GitHub repository per group
Starting Code	your group's repository from Meeting 3 (continue its `main`)

A. Practicum Outcomes

After completing this group jobsheet, you'll be able to:

1. Design the schema for Simple POS's four core tables (`categories`, `products`, `transactions`, `transaction_details`) along with their foreign key relationships.
2. Write migrations for that schema, following the golden rule "new migration, never edit an old one that's already run".
3. Write a realistic-scale seeder (300 products, 2,500 transactions) using bulk insert via `DB::table()->insert()` in batches, not `Model::create()` one row at a time, wrapping the transaction and detail inserts in `DB::transaction()`.
4. Prove an index's impact on a foreign key column with `EXPLAIN QUERY PLAN`, from `SCAN` (scanning the whole table) to `SEARCH ... USING INDEX`.
5. Replace the hardcoded product data in `TransactionController` (from Meeting 3) with a `Product` model that reads from the seeded database.

B. Preparation and Prerequisites

- **Tools**: same as Meeting 3 (PHP 8.2+, Composer, Node.js, Git), plus the `sqlite3` CLI if you have it (optional, there's a Tinker fallback if it isn't installed).
- **Git identity**: already set up since Meeting 3. If you switch laptops or use a lab machine, redo the `git config --global user.name/user.email` check from that jobsheet.
- **Continuing the code**: the group continues from its shared GitHub repository from Meeting 3, from a `main` that already contains `increment 3`. If your group's repository has problems, or Meeting 3's independent challenge isn't finished yet, start from the `simple-pos-ch03` template (`https://github.com/se-polinema/simple-pos-ch03`, created via **Use this template** as in Meeting 3's Step 1), then redo Meeting 3's Step 5 (simplifying `TransactionController`) before continuing this jobsheet, because `simple-pos-ch03` ships the production controller that calls a Model you haven't built yet.
- **Quick check** before starting:

```
git checkout main
git pull
git log --oneline -1
```

> ✓ **Checkpoint:** the top line shows the `increment 3: ...` commit.

C. Work Steps

Same workflow as Meeting 3: each build step (Step 1 through 5) happens on its own branch off `main`, gets merged via a Pull Request using **Create a merge commit** (not squash), then everyone `git pull`s before starting the next branch. Design the schema on paper before writing any migration.

Step 1: Design the schema on paper

Before writing code, the group sketches the following schema on paper or a whiteboard and agrees who will write the migration for which table:

Table	Main Columns
`categories`	`id`, `name`
`products`	`id`, `category_id` (fk), `name`, `price`, `stock`
`transactions`	`id`, `user_id` (fk), `total`, `created_at`
`transaction_details`	`id`, `transaction_id` (fk), `product_id` (fk), `qty`, `subtotal`

One `categories` row has many `products`. One `products` row can appear in many `transaction_details`, and one `transactions` row has many `transaction_details`, each recording one purchased line item. `transactions` points back to `users` (the cashier who processed that transaction, already in place since Meeting 1).

Note `transactions.total` and `transaction_details.subtotal`: both are columns that are **stored**, not recomputed every time they're read. That's a deliberate choice, because a transaction's total has to stay exactly what it was when the transaction happened, even if the product's price changes later. The validation meeting later covers why this value must also be recomputed on the server when it's saved, not just trusted from input.

> ✓ **Checkpoint:** the group has a schema sketch on paper/whiteboard, and Steps 2-5 are divided up and agreed on.

Step 2: `categories` and `products` migrations

Create a new branch off the latest `main`, e.g. `categories-products-migration`:

```
git checkout main
git pull
git checkout -b categories-products-migration
php artisan make:model Category -m
php artisan make:migration create_products_table
```

`make:model Category -m` creates the `Category` model and its migration in one command; this model is used later by the seeder in Step 4. The `Product` model is deliberately not created yet, that's part of Step 5.

Fill in the `categories` migration's `up()` method:

```
Schema::create('categories', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->timestamps();
});
```

Fill in the `products` migration's `up()` method:

```
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->foreignId('category_id')->constrained();
    $table->string('name');
    $table->unsignedInteger('price');
    $table->unsignedInteger('stock')->default(0);
    $table->timestamps();
});
```

`foreignId('category_id')->constrained()` is a pair that's often used together: the first line creates a `category_id` column as an unsigned big integer, the second adds a foreign key *constraint* pointing to `categories.id`, following Laravel's naming convention. This constraint protects data integrity, preventing a product row from pointing at a category that's been deleted, but as you'll see in Step 4, that doesn't automatically mean the column has an index.

```
git add .
git commit -m "tambah migrasi categories dan products"
git push -u origin categories-products-migration
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** `php artisan migrate:fresh` (without `--seed` yet) runs with no error, and `php artisan tinker` followed by `Schema::hasTable('products')` returns `true`.

Step 3: `transactions` and `transaction_details` migrations

Create a new branch off the latest `main`, e.g. `transactions-migration`:

```
git checkout main
git pull
git checkout -b transactions-migration
php artisan make:migration create_transactions_table
php artisan make:migration create_transaction_details_table
```

Fill in the `transactions` migration's `up()` method:

```php
Schema::create('transactions', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained();
    $table->unsignedInteger('total');
    $table->timestamps();
});
```

Fill in the `transaction_details` migration's `up()` method:

```php
Schema::create('transaction_details', function (Blueprint $table) {
    $table->id();
    $table->foreignId('transaction_id')->constrained();
    $table->foreignId('product_id')->constrained();
    $table->unsignedInteger('qty');
    $table->unsignedInteger('subtotal');
    $table->timestamps();
});
```

```
git add .
git commit -m "tambah migrasi transactions dan transaction_details"
git push -u origin transactions-migration
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** `php artisan migrate:fresh` runs with no error and lists all four new tables (`categories`, `products`, `transactions`, `transaction_details`) among the migrations it ran.

> ⚠ **If it fails:** a `no such table: categories` error while the `products` migration runs means the migration filenames are out of order, `categories`'s timestamp must be earlier than `products`'s (and `transactions` before `transaction_details`), since Laravel runs them in order by filename.

Step 4: Realistic-scale seeder and proving the index

Create a new branch, e.g. `seeder-and-index`:

```
git checkout main
git pull
git checkout -b seeder-and-index
```

Replace the contents of `database/seeders/DatabaseSeeder.php` with:

```php
<?php

namespace Database\Seeders;

use App\Models\Category;
use App\Models\User;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
```

```php
class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        $user = User::factory()->create([
            'name' => 'Test User',
            'email' => 'test@example.com',
        ]);

        $categoryIds = collect(['Makanan', 'Minuman', 'Snack', 'Lainnya'])
            ->map(fn (string $name) => Category::create(['name' => $name])->id)
            ->all();

        $products = [];
        foreach ($categoryIds as $categoryId) {
            for ($i = 0; $i < 75; $i++) {
                $products[] = [
                    'category_id' => $categoryId,
                    'name' => fake()->words(2, true),
                    'price' => fake()->numberBetween(3000, 50000),
                    'stock' => fake()->numberBetween(0, 200),
                    'created_at' => now(),
                    'updated_at' => now(),
                ];
            }
        }

        foreach (array_chunk($products, 50) as $chunk) {
            DB::table('products')->insert($chunk);
        }

        $productPrices = DB::table('products')->pluck('price', 'id');
        $productIds = $productPrices->keys()->all();

        for ($t = 0; $t < 2500; $t++) {
            DB::transaction(function () use ($user, $productIds, $productPrices) {
                $itemCount = fake()->numberBetween(1, 4);
                $total = 0;
                $details = [];

                for ($i = 0; $i < $itemCount; $i++) {
                    $productId = fake()->randomElement($productIds);
                    $qty = fake()->numberBetween(1, 3);
                    $subtotal = $productPrices[$productId] * $qty;
                    $total += $subtotal;

                    $details[] = [
                        'product_id' => $productId,
                        'qty' => $qty,
                        'subtotal' => $subtotal,
                    ];
                }

                $transactionId = DB::table('transactions')->insertGetId([
                    'user_id' => $user->id,
                    'total' => $total,
                    'created_at' => now(),
                    'updated_at' => now(),
                ]);

                foreach ($details as &$detail) {
                    $detail['transaction_id'] = $transactionId;
                    $detail['created_at'] = now();
                    $detail['updated_at'] = now();
                }
```

```
                DB::table('transaction_details')->insert($details);
            });
        }
    }
}
```

The `User::factory()->create(['name' => 'Test User', ...])` line above is identical to the user the seeder has already been creating since Meeting 1, just moved here so the whole file's contents are shown in full. `DB::table()->insert()` bypasses the Eloquent layer entirely: it builds one `INSERT` statement that writes many rows at once, far fewer round-trips to the database than calling `Model::create()` one at a time inside a loop of 300. `array_chunk()` splits the large array into smaller groups before inserting, because some database engines cap the number of rows or parameters in a single `INSERT` statement. Product prices are fetched once upfront via `pluck('price', 'id')` into an associative array, instead of being queried again for every transaction item, consistent with the reason bulk insert is used in the first place: fewer round-trips to the database means a faster seeder. `DB::transaction()` wraps the `transactions` and `transaction_details` inserts: if the process fails partway through, every change inside that block rolls back together, so the database never ends up with a transaction recorded but its details half-missing.

```
php artisan migrate:fresh --seed
```

> ✅ **Checkpoint:** the command runs with no error. Verify the row counts via `php artisan tinker`:
>
> ```
> >>> DB::table('products')->count();
> => 300
> >>> DB::table('transactions')->count();
> => 2500
> ```

Now prove the index's impact. Look at the situation **before** an explicit index exists:

```
sqlite3 database/database.sqlite \
  "EXPLAIN QUERY PLAN SELECT * FROM products WHERE category_id = 3;"
```

> ✅ **Checkpoint:** the result line includes the word `SCAN products`, meaning SQLite scans the entire table from first row to last to find a match.

> ⚠ **If it fails:** the `sqlite3` command not being recognized means its CLI isn't installed on your system. Use the Tinker alternative instead, no installation needed:
>
> ```
> php artisan tinker
> >>> DB::select("EXPLAIN QUERY PLAN SELECT * FROM products WHERE category_id = 3");
> ```
>
> The result is an associative array; its `detail` column contains the same SCAN/SEARCH text as the `sqlite3` CLI.

Create a new migration specifically for the index (don't edit an old migration that's already run):

```
php artisan make:migration add_index_to_products_and_transactions_table
```

Open the newly created migration file, then fill in its `up()` method:

```
public function up(): void
{
    Schema::table('products', function (Blueprint $table) {
        $table->index('category_id');
    });

    Schema::table('transactions', function (Blueprint $table) {
        $table->index('user_id');
    });
}
```

Run that migration. Use `php artisan migrate`, not `php artisan migrate:fresh`, so the index gets added on top of the existing seeded data instead of wiping everything again:

```
php artisan migrate
```

Run the exact same `EXPLAIN QUERY PLAN` command again as before.

> ✅ **Checkpoint:** the result line now changes to `SEARCH products USING INDEX products_category_id_index`, meaning SQLite jumps straight to the relevant rows via the index structure, without touching any other row at all.

```
git add .
git commit -m "tambah seeder skala nyata dan index foreign key"
git push -u origin seeder-and-index
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main`, pulls, and runs `php artisan migrate:fresh --seed` on their own laptop so everyone's data matches.

Step 5: Replacing the hardcoded data with a `Product` model

Meeting 3's jobsheet made a promise: the product data written directly into `TransactionController` would be replaced with a real `Product` model once the table was ready. It's time to keep that promise.

Create a new branch, e.g. `product-model`:

```
git checkout main
git pull
git checkout -b product-model
php artisan make:model Product
```

Change the `create()` method in `app/Http/Controllers/TransactionController.php`, from returning a hardcoded `collect([...])` to:

```php
use App\Models\Product;

// ...

public function create()
{
    $products = Product::take(12)->get();

    return view('pos.create', ['products' => $products]);
}
```

Delete the hardcoded `collect([...])` array along with the now-unused `use` it depended on. The `store()`, `index()`, and `show()` methods stay as they were in Meeting 3 for now, full listing with pagination is covered in the upcoming ORM & relations meeting, so `take(12)` here is just a temporary display limit.

```
git add .
git commit -m "ganti data hardcoded dengan model product"
git push -u origin product-model
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** opening `http://127.0.0.1:8000/pos` shows 12 seeded products (with random names and prices from `fake()`), no longer the six hardcoded products from Meeting 3.

> ⚠ **If it fails:** a `Class "App\Models\Product" not found` error means `use App\Models\Product;` wasn't added at the top of the controller, or `php artisan make:model Product` hasn't been run/committed.

Step 6: Together, integration test and code review

Everyone runs `git checkout main && git pull`, `php artisan migrate:fresh --seed`, then repeats the `EXPLAIN QUERY PLAN` test and opens `/pos` on their own laptop. After that, the group reads through the code together: each member explains one migration or part of the seeder that they **didn't** write themselves.

> ✅ **Checkpoint:** the `EXPLAIN QUERY PLAN` result and the `/pos` page look exactly the same on every member's laptop.

Step 7: Group independent challenges and the `increment 4` commit

Split the following tasks among the group's members, so each member gets at least one meaningful commit recorded through their own Pull Request:

- Add an index on the `transaction_details.product_id` column via a new migration, then prove it with `EXPLAIN QUERY PLAN` on a query that looks up every detail row for a specific product.
- Add a new `sku` column (string, unique, nullable) to the `products` table via a new migration, not by editing an old migration that's already run.
- Add an `is_active` column (boolean, default `true`) to `products` via a new migration, then update the seeder so about 10% of products are created with `is_active` set to `false`.
- Write one new listing query (e.g. transactions within a date range) via `php artisan tinker`, then run `EXPLAIN QUERY PLAN` on that query to check whether it's already using an existing index.
- Change how many products `TransactionController::create()` shows, from `take(12)` to showing every product whose `stock` is above 0.

For each task: create a new branch (e.g. `index-transaction-details`), do the work, commit, push, then open and merge its Pull Request (follow Meeting 3's Step 3 workflow).

Once every Pull Request is in and merged, one member closes out the group's work:

```
git checkout main
git pull
git commit --allow-empty -m "increment 4: skema, seeder, dan index simple pos"
git push
git log --pretty="%h %an %s"
```

> ✓ **Checkpoint:** the top line of `git log --pretty="%h %an %s"` shows `increment 4: ...`, and the lines below it show every group member's name.

D. Deliverables

Submit the following in the format your instructor requests:

- Link to the group's GitHub repository.
- Screenshot of the `EXPLAIN QUERY PLAN` output before the index (`SCAN`) and after it (`SEARCH ... USING INDEX`).
- Screenshot of the `/pos` page showing seeded products (no longer hardcoded data).
- Output of `git log --pretty="%h %an %s"` showing at least one commit per member.
- A task-division table: member name | step/task done | commit hash.

E. Grading Rubric

Component	Weight	Full Marks (100%)	Minimum Marks
Work steps completed (group)	40%	Steps 1-7 done, seeder runs with the correct row counts, `/pos` shows data from the database	Most steps done, seeder and `/pos` work
Checkpoints verified (group)	25%	`EXPLAIN QUERY PLAN` screenshots before/after, task-division table, and a complete, correct git log	Some checkpoints proven
Per-member contribution (individual)	25%	At least one meaningful commit under each member's name, matching the task-division table	Commits exist but are small or their relevance is unclear

Component	Weight	Full Marks (100%)	Minimum Marks
Repository and commit hygiene	10%	The `increment 4` message is exact, new migrations (not edits to old ones), no `vendor/`/`node_modules/`/`.env` included, PRs merged cleanly (not squashed)	Commits exist, but the message is messy or a PR was squashed