There are numerous benefits of having a standardized local development environment for your entire team to work on and share. This arrangement is essential for reducing bugs and interchanging team members amongst projects – a topic discussed at length in Vagrant Up Can Make Development Easier.

One of the major milestones to achieve in your team's workflow is assuring everyone is working on the same version of your back-end. This is most often a database of some kind. Whether NoSQL, relational (MySQL, SQL Server), or serverless, we recommend standardizing and versioning your data to ensure the greatest efficiency.

Typical best practices require that each developer has his/her own copy of the database to work. This avoids imposing bugs or changes on other developers which causes unnecessary work, and removes unnecessary roadblocks and conflicts. It's generally the best overall approach to an Agile workflow.

Migrations

Nowadays most development frameworks and platforms have some form of *migrations*. WordPress is a notable exception, although some hybrid solutions do exist via plugins like WP DB Migrate.

Wikipedia defines a migration as "the management of incremental, reversible changes to relational database schemas."

In practice this means that regardless of which branch of code the developer is working on, a single command can be executed to create, build, and/or update the database so that work can begin immediately. This almost completely alleviates the need for copying/importing databases, fixing conflicts, or dealing with connecting to remote databases and the inevitable connectivity issues that arise.

For example, using Laravel the process would be one or two simple commands:

```
# Create the articles table
php artisan make:migration create_articles_table --create=articles

# Add to the existing articles table
php artisan make:migration add_votes_to_articles_table --
table=articles
```

At this point, every branch of code you go to work on will have a fresh database. Any schema changes necessary to complete your work could be built into a fresh migration script. This can be executed on top of your existing schema.

When merged into your primary release branch, any future development will have your migration(s). As a result, the workflow continues uninterrupted.

Example Migration

Below is an example migration to set up a simple table using Eloquent (Laravel):

```
<?php

use IlluminateDatabaseSchemaBlueprint;
use IlluminateDatabaseMigrationsMigration;

class CreateArticlesTable extends Migration
{

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('articles', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title');
            $table->string('body');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */

    public function down()
    {
        Schema::drop('articles');
    }

}
```

Seeding

Seeding is the process of populating data into a database. This can be “dummy” data or enum data (or both), and is very useful for testing and tuning your models; particularly, in object oriented frameworks.

A simple example that generates a single article:

```
<?php

use IlluminateDatabaseSeeder;
use IlluminateDatabaseEloquentModel;

class DatabaseSeeder extends Seeder
{

    /**
     * Run the database seeds.
     *
     * @return void
     */

    public function run()
    {
        DB::table('articles')->insert([
            'title' => str_random(10),
            'body' => str_random(1000)
        ]);
    }

}
```

For further reading on other Development Principles and Best Practices, check out Dev Principle #1: Choose Appropriate Variable Names and Dev Principle #2: Coding Layout and Style Matter.

This was originally posted at Fresh Consulting.