**DATABASE MIGRATION**

We can migrate a Database in an existing DB. We have two separate instances: Staging (Development) and Production. We will work on the development and once everything works then will deploy code on production.

During the development we can create a new collection, change the existing column, update the content etc. We have to manage all things and need to deploy on production, so we have to follow the below ways-

1. Using the <https://docs.strapi.io/dev-docs/database-migrations> document (npx strapi generate)
2. Using the <https://knexjs.org/guide/migrations.html#migration-cli> document

In both cases, we have to use the **Knex** SQL builder. It also supports other databases as well.

**Option 1 -**

We can go follow this approach with below steps-

1. Need to run default command

Npx strapi generate

Need to select migration option and name of the migration

1. It will create one file under the “database/migrations” folder
2. It also create a entry in database table “strapi\_migrations”
3. Now whenever we build the project, newly created migration files run on that time. We don’t need to run manually.

In Above approach, whenever we run the project. First it will run migration then run strapi.

**Challenge-** I have noticed some of the migrations not updating the database fields. Strapi team looking on that.

**Option 2 -**

We have to follow the below steps for option #2-

1. Need to navigate on your strapi project on cli
2. Install the Knex using command

npm install knex -g

1. knex init

This command will create one **knexfile.js** file. In that file all database connections will be there. We need to put necessary information in that. It also creates one table “knex\_migrations” & “knex\_migration\_lock” in the database. Its table holds all migration entries.

1. knex migrate:make migration\_name

This command will generate a migration file under the “migration” folder on root. We can create as many files as you want.

1. We can execute migration using

knex migrate:latest

1. Whenever we create a migration file, once

**Knexfile.js Content-**

module.exports = {

development: {

client: 'postgresql',

connection: {

database: 'regular\_test',

user: 'root',

password: 'root'

},

pool: {

min: 2,

max: 10

},

migrations: {

tableName: 'knex\_migrations'

}

},

**Migration File content-**

/\*\*

\* @param { import("knex").Knex } knex

\* @returns { Promise<void> }

\*/

exports.up = function (knex) {

return knex.schema.createTable('abctests', function (table) {

table.increments();

table.string('name');

table.timestamps();

})

// console.log("Table Created") //- NOT WORKING

};

/\*\*

\* @param { import("knex").Knex } knex

\* @returns { Promise<void> }

\*/

exports.down = function (knex) {

};

**Challenge-** We have to run the “knex migrate:latest” command every time. I also noticed one thing when we rebuilt the project so at that time previously created tables “knex\_migrations” & “knex\_migration\_lock” both had been deleted. It also removes previously created database tables and content of that table. But migrations files are still there, so in that case all migrations run again. So it’s risky for us.

**Conclusion-**

Now we have 2 options, we can go to any of them.The Strapi team is also working on the above challenges or we also need to find some solutions from our side.

Currently we have found some easiest way to achieve that. Either with the CMS side or migration. Please review the below process, which mention all easiest possible way.

**Process -**

1. **Create New Table** -

Whenever we need to create a **new collection type**. We will create from the local CMS version. Then we can push the code and the table will be created into the Live Database. Don’t need to create a separate migration file.

**Status-** **Done**

1. **Rename Existing Table -**

If we want to modify the collection (database table) name then the easiest way is to directly modify the schema file.

We have to update the schema.json file under the particular collection type folder on the path “gft-strapi-consumer/src/api”. We can update the value of “collectionName” in the schema.json file. After that we have to build the project or need to run the “npm run develop” command. Now the database table name will change but your collection type folder and all things remain the same.

If you also want to change the folder name, same like as a table name then need to follow below steps-

* Renaming folder under api
* Rename of the files under that folder
* Search and replace references of the old component name in that folder in all files like controller, route etc
* Done a strapi build develop
* Run npm run develop and the interface looks good

If you only want to change the display name then directly modify it from the CMS or change the value of “displayName” field under the schema.json file.

**Status-** **Done**

1. **Drop Existing Table**

We can do it either by creating a migration script or directly from the CMS. Best approach is that we can remove it from the CMS and its data can also be removed.

**Status-** **Done**

1. **Add New Column in existing table-**

We can do it either by creating a migration script or directly from the CMS. Best approach is that we can add directly from the CMS.

**Status-** **Done**

1. **Rename Column in existing table-**

We can change the column name from the CMS or using the migration command. But in both the cases your existing **data will lose** from that field.

Strapi team working on that issue - <https://docs.strapi.io/dev-docs/database-migrations>

We can create migration file using this command “

npx strapi generate” OR knex migrate:make abctests

**Status-** **Done**

1. **Delete Column -**

We can do it either by creating a migration script or directly from the CMS. Best approach is that we can directly delete from the CMS. Your data will be lost in both cases.

**Status-** **Done**

1. **Add new Content**

We can directly add from CMS.

**Status-** **Done**

1. **Edit Content**

We can directly edit from CMS.

**Status-** **Done**

1. **Delete Content**

We can directly delete from CMS.

**Status-** **Done**

1. **Seeder**

**Status-** I checked all related features regarding the seeder but it was only for testing purposes.

………………………………………………………………………………………………………

**Export / Import**

<https://docs.strapi.io/dev-docs/data-management>

Occasionally, you need to move data out of or into a Strapi instance. This is possible with the data management system which uses CLI-based commands:

* Use [strapi export](https://docs.strapi.io/dev-docs/data-management/export) to create a data backup, for archive purposes or to import it in another instance.
* Use [strapi import](https://docs.strapi.io/dev-docs/data-management/import) to restore data from a backup.
* Use [strapi transfer](https://docs.strapi.io/dev-docs/data-management/transfer) to transfer data between local and/or remote instances.

<https://docs.strapi.io/dev-docs/data-management/transfer>

The strapi transfer command streams your data from one Strapi instance to another Strapi instance. The transfer command uses strict schema matching, meaning your two Strapi instances need to be exact copies of each other except for the contained data. The default transfer command transfers your content (entities and relations), files (assets), project configuration, and schemas. The command allows you to transfer data:

* from a local Strapi instance to a remote Strapi instance
* from a remote Strapi instance to a local Strapi instance