Creating mass data for testing with AditoFaker

AID120

ADITO Academy

Version 1.3 | 25.10.2022

Versions	Changes
1.0	Release Version
1.2	Changed the faker folder structure in a ADITO Project
1.3	Added scripts and links for the manual setup =======

Index

Character Formatting

The following signs will point you to specific sections:

Hints and notes.

Tips and tricks.

This is important!

Warning! These actions are dangerous and can result in data loss!

The following font formatting applies:

Font type	Meaning
Mask	The mask, table or button to which the section refers
"Mask"	Terms that originate from the system and that need to be emphasized in the reading flow
`code();`	Code and program parts

1. What is Faker?

Faker is a widely known module used to generate many different types of fake data for testing and other purposes. There are various language ports of Faker including PHP, Perl, Phyton, JavaScript, Java and more.
Within the ADITO Context we created a Node.js package (AditoFaker) based on Faker.js and Sequelize (for database handling) to generate massive amounts of faked data for testing purposes within ADITO.

2. How to set up AditoFaker

2.1. Installation

First: AditoFaker is a development tool! You can't generate fake data from the client as an administrator. In any case you need access to the ADITO Designer, your systems database and you need to know how to configurate and execute AditoFaker properly.

2.1.1. Setup before installation

To work with AditoFaker you need the following setup:

- ADITO Designer of version 2022.0 or newer. It should work work 2021.1.0 as well but it's implemented and tested from 2022.0 upwards. For any version below 2022.0 its functionallity cannot be garantueed.
- Installed Node.js plugin
- ADITO xRM of version 2022.2.0 onwards. If your project has a older version you need to make some changes manually.
- Access to your database: active connection, host, port, user, password

You can use faker for every database / adito version if you use it without the ADITO Designer. In this case you need to install node.js manually and use it within a different IDE (like VS Code) or at the command line.

2.1.2. Install AditoFaker

If you have a xRM of version 2022.2.0 or upwards or you fulfilled all steps of the manual setup, you can just execute `npm install` within your designer and AditoFaker will get installed.

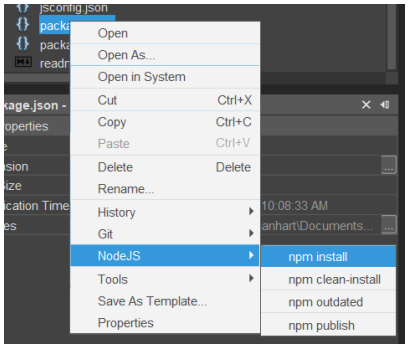


Figure 1. ADITO Designer npm install

If you're not sure if you have the correct xRM version please check within your project if you have a package.json file and if you have the AditoFaker dependency within the json.

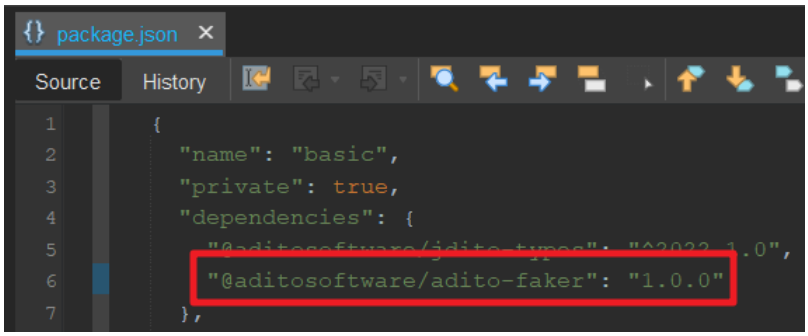


Figure 2. ADITO Designer package.json add faker dependency

```
"@aditosoftware/adito-faker": "^1.1.1"
```

faker dependency in package.json (make sure you use the most recent version)

Now all you need to start to configure AditoFaker and generate data is installed.

2.2. Initial configuration

Before you can use AditoFaker you need to make your initial configuration for your project. This means you need to apply your database structure to AditoFaker and set up the configuration how to fake the different fields.

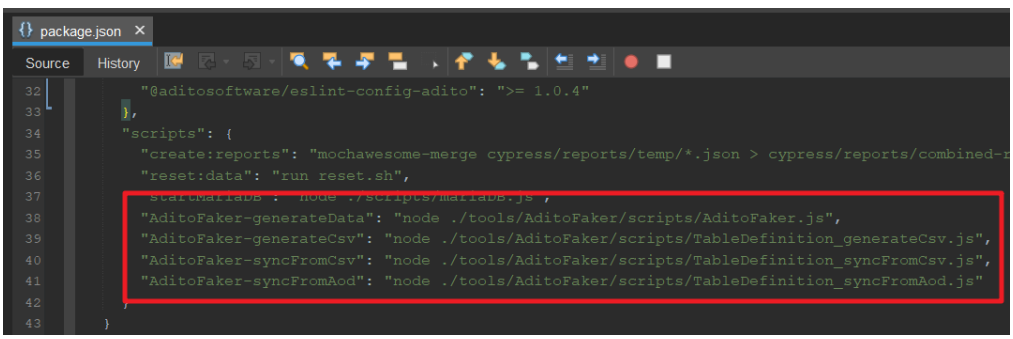
Currently initial configurations come bundled with xRM Basic (>= 2022.2.0) for a few tables.

> In the future these default configurations will be extended to all main tables and modules. So you only have to add your project specific tables in the future.

All configurations of AditoFaker are done within a **AditoFaker** folder within the `tools` folder in your project (Note: you can create the tools folder if it doesn't exist already).

The initial configuration of AditoFaker contains the following within your ADITO project:

- A folder **AditoFaker** within the `tools` folder of your project, it contains the following folders and files:
  - **configurations**
    - **configuration.json**: Contains the executing configuration of AditoFaker
    - **tableDefinition.json**: Contains the database structure with the configuration which data to generate
    - **tableDefinition.csv**: Not mandatory. It's a simplified variant of tableDefinition.json for easier editing
  - **scripts**
    - **AditoFaker.js**
    - **TableDefinition_generateCsv.js**
    - **TableDefinition_syncFromAod.js**
    - **TableDefinition_syncFromCsv.js**

> These scripts need to be configurated within the scripts param in the **package.json**.
>
> 
>
>
> *Figure 3. ADITO Designer package.json scripts to add*

If you have a xRM of version 2022.2.0 or newer, you should have already all neccessary files and changes within your project.
If this is not the case check appendix C: Set up AditoFaker without the preset of xRM 2022.2.0

```
"AditoFaker-generateData": "node ./tools/AditoFaker/scripts/AditoFaker.js",
"AditoFaker-generateCsv": "node ./tools/AditoFaker/scripts/TableDefinition_generateCsv.js",
"AditoFaker-syncFromCsv": "node ./tools/AditoFaker/scripts/TableDefinition_syncFromCsv.js",
"AditoFaker-syncFromAod": "node ./tools/AditoFaker/scripts/TableDefinition_syncFromAod.js"
```

faker scripts in package.json (make sure you use the most recent version)

If you have all this files and installed all npm modules properly you should see the scripts within the execute window in the ADITO designer:

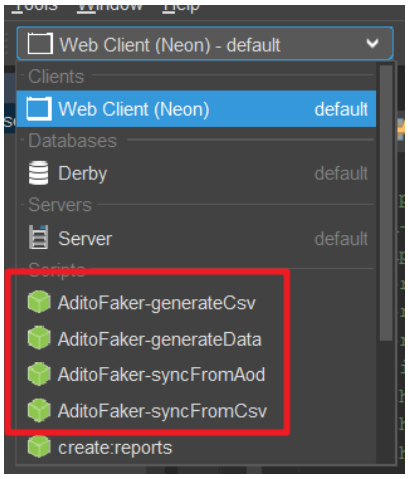


Figure 4. ADITO Designer executable scripts

2.3. Sync the TableDefinition with the Data_alias

AditoFaker uses the TableDefinition (tools/AditoFaker/configurations/tableDefinition.json) to get the database structure and configuration for each table and field.
So the first step to set up faker is to generate or sync the projects TableDefinition with the Data_alias.aod file of the project. For this purpose we have the script **AditoFaker-syncFromAod**. If you execute it, it will read your projects Data_alias.aod file and sync it with the TableDefinition.
Missing tables and fields get removed and new tables and fields will be added.

If a table has already a existing faker configuration within the TableDefinition, it

wont get lost with **AditoFaker-syncFromAod**.

If you want to use a alias file different then the default Data_alias.aod see change paths in configuration

Now you have your TableDefinition synced with your alias and you are ready to go. If you don’t want to edit the JSON file see the next part: **Generating the TableDefinition as CSV file**

2.4. Generating the TableDefinition as CSV file

To make the configurational work with AditoFaker easier, you can generate and edit a CSV file instead of the TableDefinition JSON.
In this case we have the script **AditoFaker-generateCsv**. Execute this script and you will recieve your **TableDefinition.csv** file your projects tools/AditoFaker/configurations/ folder.

If you want to work with the CSV file make sure you use LibreOffice Calc and select the semicolon **;** as field limiter and a single quotation mark **'** as field limit.
Otherwise you can get problems with your CSV file. A simple text editor like Notepad++ will work as well.

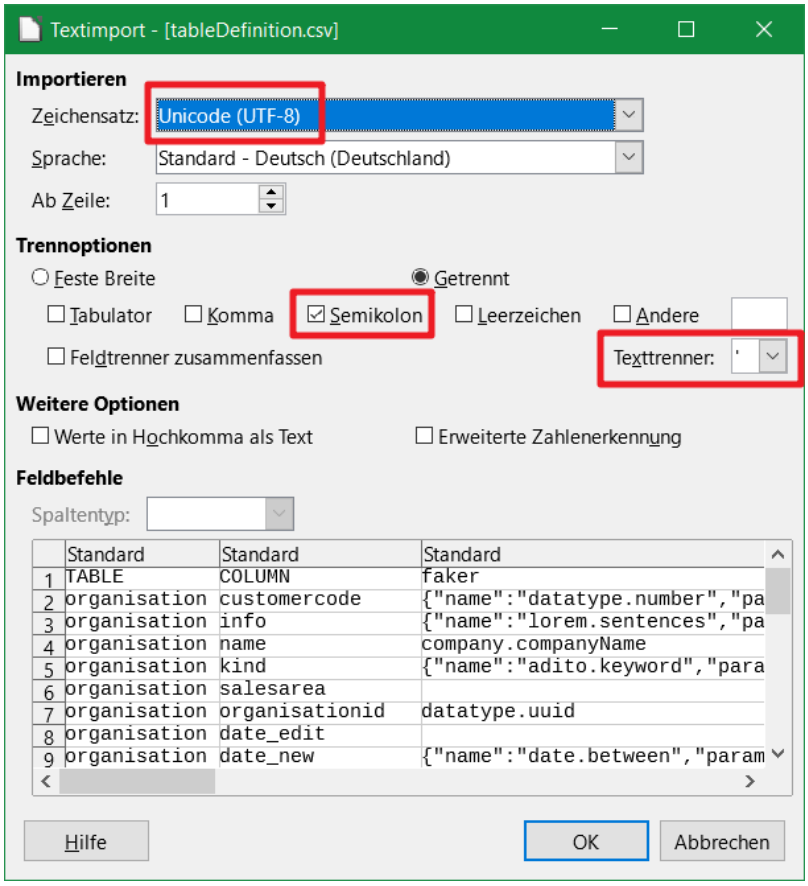


Figure 5. LibreOffice Calc open CSV

Now you can see all your tables and columns with their faker configuration.

2.5. Configure the TableDefinition

Within the TableDefinition we configure the default values how faker should fill the database fields.

	A	B	C	D	E
1	TABLE	COLUMN	faker	relation	value
2	organisation	customercode	{"name":"datatype.number","params":[{"min":1000,"max":999999}],"unique":"ORGANISATION.CUSTOMERCODE"}		
3	organisation	info	{"name":"lorem.sentences","params":[10]}		
4	organisation	name	company.companyName		
5	organisation	kind	{"name":"adito.keyword","params":["OrganisationType"]}		
6	organisation	salesarea			
7	organisation	organisationid	datatype.uuid		
8	organisation	date_edit			
9	organisation	date_new	{"name":"date.between","params":["2019-01-01T00:00:00.000Z","2022-01-06T00:00:00.000Z"]}		
10	organisation	user_new			faker
11	organisation	user_edit			
12	organisation	picture			
13	contact	isolanguage			deu
14	contact	contactid	datatype.uuid		
15	contact	organisation_id		organisation.organisationid	
16	contact	status	{"name":"adito.keyword","params":["ContactStatus"]}		
17	contact	person_id		person.personid	
18	contact	address_id	datatype.uuid		
19	contact	contactrole	{"name":"adito.pickFromDb","params":["AB_KEYWORD_ENTRY.TITLE","AB_KEYWORD_ENTRY.AB_KEYWORD_CATEGORY_ID = '866fd65f-131c-4038-99ce-78606971ec98' and AB_KEYWORD_ENTRY.ISACTIVE = '1'"]}		
20	contact	department	{"name":"adito.pickFromDb","params":["AB_KEYWORD_ENTRY.TITLE","AB_KEYWORD_ENTRY.AB_KEYWORD_CATEGORY_ID = 'cda5deac-5b7c-4c98-9904-2eb3b224235d' and AB_KEYWORD_ENTRY.ISACTIVE = '1'"]}		
21	contact	contactposition	{"name":"adito.pickFromDb","params":["AB_KEYWORD_ENTRY.TITLE","AB_KEYWORD_ENTRY.AB_KEYWORD_CATEGORY_ID = 'bfc9eb87-0292-4fa1-9c87-39666c2ae6d2' and AB_KEYWORD_ENTRY.ISACTIVE = '1'"]}		
22	contact	date_edit			
23	contact	user_edit			
24	contact	date_new		organisation.date_new	
25	contact	user_new			faker
26	contact	lettersalutation			

Figure 6. Example of a TableDefiniton in LibreOffice Calc

For the faker value we have three possible properties: **faker**, **relation** and **value**

You need to set just one of these three for each field, if multiple properties are set just the first one will

get used (ordered as shown above).

2.5.1. TableDefinition faker property

The faker property is the most powerful property to generate fake data. With this you have access to all available functions of the current faker api.
See fakers API documentation for more information about available functions and their usage. In addition to fakers default functions we built some AditoFaker specific functions.
See Appendix B: ADITO specific faker functions for more informations about the adito functions (usage is the same as the default faker functions).

There are different ways of using the faker property depending of the function and needed functionality:

2.5.1.1. Call a faker function without params

If you want to call a simple faker function without params you can just set the function name (without faker.) in the faker property.
This is the simplest way of calling a function to fake data. All other configurations of the faker property need a JSON object.

TABLE	COLUMN	faker
organisation	name	company.companyName

Figure 7. TableDefinition: set a simple faker function without params

2.5.1.2. Call faker functions with a extended configuration

All configurations of faker functions are done within a JSON object. So all of the following descriptions expect a JSON with this format:

```
{
  "name": "modulename.functionname",
  "params": ["param1", "param2", "paramN"],
  "unique": "TABLENAME.FIELDNAME"
}
```

2.5.1.2.1. Faker configuration property: name

The **name** property is required if you want to configure faker with a JSON object. It must contain the name of the faker function.

2.5.1.2.2. Faker configuration property: params

The **params** property is optional. With this property you can provide params to the faker function (2d array, params will be passed like given in the array).

TABLE	COLUMN	faker
organisation	info	{"name":"lorem.sentences","params":[10]}

Figure 8. TableDefinition: set a faker function with a param

2.5.1.2.3. Faker configuration property: unique

The **unique** property is optional and a AditoFaker specific implementation. You can set a TABLE.FIELD from which you want to select the values and prevent faker from generating a value twice.
All distinct values if the given field will be selected and stored during execution of faker. Each newly generated value will get stored as well.
To store and check the unique values the faker function faker.helpers.unique is used. A good example for the unique param is ORGANISAITON.CUSTOMERCODE → prevent faker from generating a customercode twice.
If you want to load the unique values from the same field where you use the function you dont need to provide a TABLE.FIELD, just set a boolean true as value for the unique property.

If the faker unique function generates a value thats still exists, faker will retry the data generation 10 times before running into a error.

TABLE	COLUMN	faker
organisation	customercode	{"name":"datatype.number","params":[{"min":1000,"max":999999}],"unique":"ORGANISATION.CUSTOMERCODE"}

Figure 9. TableDefinition: use the unique property

2.5.2. TableDefinition property: relation

With the relation property you can relate to a field from your current dataset or a related dataset (needs to be processed before the relation itself).
With this property you can share values like UIDs between fields and tables during data generation.

TABLE	COLUMN	faker	relation
contact	date_new		organisation.date_new

Figure 10. TableDefinition: fill a field with a related value

See relations with faker to know how to relate tables

2.5.3. TableDefinition property: value

A value is the simplest way of filling data with AditoFaker. You can set a fixed value for the field and it

will be written to the database.

TABLE	COLUMN	faker	relation	value
contact	user_new			faker

Figure 11. TableDefinitoon

2.6. Sync the configurated CSV file back to JSON

If you have edited and saved your CSV file you can sync it back to the JSON with the script **AditoFaker-syncFromCsv**.
Make sure you didn't change the CSV field limiter and limit, if you did you will recieve a error by parsing the CSV file.

3. Execute AditoFaker

Now your TableDefiniton is configurated and you're good to go.
You just need to configure the TableDefinition once at start, if you have new tables / columns, if you have changed tables / columns or if you want to change the initial faker configuration.

3.1. Set up configuration.json

The next step is to set up faker to execute and generate data. This is done with a **configuration.json** in your projects tools/AditoFaker/configurations/ folder.

The configuration.json can contain these main properties:

```
{
  "locale": "de",
  "dbConnection": "mariadb://dbuser:dbpassword@localhost:3306/aditodata",
  "data": {},
  "progressLog": false,
  "sequelizeLog": false
  "paths": {}
}
```

3.1.1. configuration.json property: locale

The **locale** property is optional and contains a locale to use faker in different languages.
To see the available locales check fakers localization table.

3.1.2. configuration.json property: dbConnection

The **dbConnection** property is required and contains the connection string for the database.
At the example you can see a connection string for mariadb. Make sure that you reach the given database.

If you want to connect to other databases than mariadb please check the sequelize documentation.

If you want to connect AditoFaker to a SSP cloud database you need to open a tunnel to the system with the designer or the tunnel.bat from the SSP and connect AditoFaker to localhost.

Currently just the sequelize connection via connection string is implemented, its planned that other variations like configuraiton via JSON will be added in the future.

3.1.3. configuration.json property: data

The **data** propety is required and the most important property of the configuration.
It contains all informations about which data should be generated.

It's a JSON with key: value for each table object to generate data for.

3.1.3.1. data object

A data object configuration represents a single object or a related bunch of objects to generate data for.
You can apply the following properties recursivly for all data objects.

Just add a tablename as key to the data object property and a table object configuration as value.

Example data object

```
{
  "data": {
    "table1": {},
    "table2": {},
    "table2$alias": {},
    "tableN": {}
  }
}
```

If you want to generate multiple different sets of data for the same table you can set the same table multiple times as key with a alias.
Just add the alias after the tablename with a $ sign. You can relate to the table by

> adding the alias.

3.1.3.2. table object

A table object configuration can contain the following main properties:

```
{
  "count": 1,
  "excludedFields": ["field1", "field2", "fieldN"],
  "specialFields": {},
  "relations": {}
}
```

3.1.3.2.1. table object property: count

The **count** property is optional and default 1. It contains the count of datasets to generate for the single table.
Instead of a fixed number you can set a array with to numbers as count between as well.

For example **"count": [1, 10]** will generate a amount between 1 to 10 datasets for each execution or related dataset.

3.1.3.2.2. table object property: excludedFields

The **excludedFields** property is optional and contains a array of fields to skip from the TableDefiniton.
If you want to prevent faker from filling all configurated fields of the TableDefinition just add them to the specialFields and they will get skipped.

3.1.3.2.3. table object property: specialFields

The **specialFields** property is optional and contains a JSON object with faker configurations for single fields of the table.
If you want to overwrite the faker configuration of the TableDefinition just for this single table or execution use this property.

You can configure the fields one to one like in the TableDefinition.

Example specialFields object

```
{
  "field1": {
    "value": "value1"
```

```
  },
  "field2": {
    "relation": "table.field"
  },
  "field3": {
    "faker": "fakerModule.functionName"
  },
  "fieldN": {
    "faker": {}
  }
}
```

3.1.3.2.4. table object property: relations

The **relations** property is optional and contains related table objects.
You can add related table objects recursivley in exact the same way than adding table objects to the main data object.

Within a related object you have access to all fields of all upward parent datasets.
That means if you have the organisation in the main data object, the contact related to organisation and the person related to contact you have access to the fields of organisation and contact within the person configuration.

If you want to add multiple table objects for the same table to a parent configuration you can do this like in the main data object by setting a alias for a table.
Example: "address$office" & "address%home" → if you want to relate to one of these you can do this by relating "address%home.address".

By adding related table objects pleas keep in mind that the count of datasets will be generated for each parent dataset.

This can extend to massive amounts of data to generate.

Example main data object with a table object for organisation and a related table object for contact

```
"data": {
    "organisation": {
        "count": 10,
        "excludedFields": ["salesarea", "picture", "date_edit", "user_edit"],
        "relations": {
            "contact": {
                "excludedFields": ["person_id", "lettersalutation", "date_edit", "user_edit", "contactrole", "department",
"contactposition"]
            }
        }
    }
}
```

This configuration will generate 10 organisations with a contact dataset for each of them.

3.1.4. configuration.json property: progressLog

The **progressLog** property is a optional boolean (default: false). If true you will get logs of the progress of AditoFaker for each called insert batch (50k datasets) and each finished insert batch.

```
FakerAdito: call insert 50000 datasets for activity
FakerAdito: call insert 50000 datasets for activitylink
FakerAdito: inserted 50000 datasets  for 'activitylink'
```

Figure 12. Example progress log

If you generate massive amounts of data (> 1.000.000) its recommended to activate the progress log to see if and when the database slows down with inserting datasets.

3.1.5. configuration.json property: sequelizeLog

The **sequelizeLog** property is a optional boolean (default: false). If true you will recieve all database logs of sequelize (statements, inserts, errors, etc.)

```
Executing (default): SELECT 1+1 AS result
Executing (default): select distinct organisation.customercode from organisation where organisation.customercode is not null
Executing (default): SELECT `keyid` FROM `ab_keyword_entry` AS `ab_keyword_entry` WHERE `ab_keyword_entry`.`ab_keyword_category_id` IN (SELECT `ab_keyword_categoryid` FROM `ab_keyword_category` W
HERE `ab_keyword_category`.`name` = 'OrganisationType') AND `ab_keyword_entry`.`isactive` = 1;
Executing (default): SELECT `keyid` FROM `ab_keyword_entry` AS `ab_keyword_entry` WHERE `ab_keyword_entry`.`ab_keyword_category_id` IN (SELECT `ab_keyword_categoryid` FROM `ab_keyword_category` W
HERE `ab_keyword_category`.`name` = 'ContactStatus') AND `ab_keyword_entry`.`isactive` = 1;
Executing (default): SELECT `keyid` FROM `ab_keyword_entry` AS `ab_keyword_entry` WHERE `ab_keyword_entry`.`ab_keyword_category_id` IN (SELECT `ab_keyword_categoryid` FROM `ab_keyword_category` W
HERE `ab_keyword_category`.`name` = 'ActivityDirection') AND `ab_keyword_entry`.`isactive` = 1;
Executing (default): SELECT `keyid` FROM `ab_keyword_entry` AS `ab_keyword_entry` WHERE `ab_keyword_entry`.`ab_keyword_category_id` IN (SELECT `ab_keyword_categoryid` FROM `ab_keyword_category` W
HERE `ab_keyword_category`.`name` = 'ActivityCategory') AND `ab_keyword_entry`.`isactive` = 1;
Executing (default): select distinct contact.contactid from contact where contact.contactid is not null and (CONTACT.PERSON_ID is not null)
Executing (default): select distinct salutation.salutation from salutation where salutation.salutation is not null and (SALUTATION.ISOLANGUAGE = 'deu')
Executing (default): SELECT `keyid` FROM `ab_keyword_entry` AS `ab_keyword_entry` WHERE `ab_keyword_entry`.`ab_keyword_category_id` IN (SELECT `ab_keyword_categoryid` FROM `ab_keyword_category` W
HERE `ab_keyword_category`.`name` = 'PersonGender') AND `ab_keyword_entry`.`isactive` = 1;
Executing (default): select distinct ab_keyword_entry.title from ab_keyword_entry where ab_keyword_entry.title is not null and (AB_KEYWORD_ENTRY.AB_KEYWORD_CATEGORY_ID = '866fd65f-131c-4038-99ce-
78606971ec98' and AB_KEYWORD_ENTRY.ISACTIVE = '1')
Executing (default): select distinct ab_keyword_entry.title from ab_keyword_entry where ab_keyword_entry.title is not null and (AB_KEYWORD_ENTRY.AB_KEYWORD_CATEGORY_ID = 'cda5deac-5b7c-4c98-9904-
2eb3b224235d' and AB_KEYWORD_ENTRY.ISACTIVE = '1')
Executing (default): select distinct ab_keyword_entry.title from ab_keyword_entry where ab_keyword_entry.title is not null and (AB_KEYWORD_ENTRY.AB_KEYWORD_CATEGORY_ID = 'bfc9eb87-0292-4fa1-9c87-
39666c2ae6d2' and AB_KEYWORD_ENTRY.ISACTIVE = '1')
```

Use the sequelize log just to debug errors. Because of the huge amounts of inserts it can get very big.

3.1.6. configuration.json property: paths

The **paths** property is a optional JSON object to change the default paths for the Data_alias.aod and the tableDefinition.json.

Example paths object

```
{
  "aod": "./your/file/path/Data_alias.aod",
    "tableDefinition": "./your/file/path/tableDefinition.json"
}
```

- **aod**: contains the path to the alias aod file
- **tableDefinition**: contains the path to the TableDefinition

If you use the ./ path you will start in your projects home path where you execute AditoFaker.

3.2. Generating massdata with AditoFaker

Now with a proper configuration.json you can execute AditoFaker with the **AditoFaker-generateData** script in the designer.
AditoFaker will start, test the database connection and set up all it needs to generate the data.
Then the data object will get evaluated, the amount of data to generate for each object estimated and printed to the console.

Example start execution message of AditoFaker

```
FakerAdito: initialized configuration. Estimated datasets to generate: 299.500:
-> organisation: 1.000
-> contact: 13.500
-> activity: 108.750
-> activitylink: 108.750
-> address: 14.500
-> communication: 40.500
-> person: 12.500
Database connection has been established successfully.
FakerAdito: start generating data.
```

Now AditoFaker starts generating the data. If a table has reached 50.000 generated datasets its provided to a insert que with 4 threads for database inserting.
Each thread will open 5 connections to the database and send 5 inserts with 10.000 datasets each. The whole data generation and insertion work is done asynchrounus within different node threads.

Once AditoFaker finished generating the data it will print this information to the console. That means the data generation has finished and all inserts are qued and waiting for execution.
When everything is finished you will get a finish message with the time it took to execute the data generation and the true amount of data generated (it can differ from the estimated counts cause you can configure random counts).

Example finish execution message of AditoFaker

```
FakerAdito: finished generating data, waiting for insert to finish.
FakerAdito: finished execution after: 00:01:00.
Inserted 298.118 datasets:
-> activity: 108.125
-> activitylink: 108.125
-> organisation: 1.000
-> contact: 13.478
-> address: 14.478
-> communication: 40.434
-> person: 12.478
```

Be careful with generating massive amounts of data (multiple millions at once).
During tests we encountered Problems with our standard SSP mariadb at about 7 million datasets.
At this mark it took us very long for each batch to insert (15 minutes for 10.000 datasets and more). We will update this manual if we have more detailed informations about the problem.

Take care of your RAM capacity with complex relation trees. The more relations you have per table the longer it takes to generate a 50.000 datasets batch for the very first table objects.
This means there is a lot of data to store temporary in the RAM. It is better to execute a smaller configuration more times than a big one at once.
Maybe we add a dynamic configurable batch size for the table objects in the future if we encounter some problems in this topic.

3.3. Benchmark

Within our tests at a local pc within VS Code and a standard SSP mariadb connected via SSH we managed to generate between 50.000 and 300.000 datasets per minute.
The more we generated at once and the bigger the database got the longer it tooks to insert the data.
The numbers we recieved were tested with a fresh startet and nearly empty database.
The amount of data you can generate and insert depends on the complexity of your data configuration as well.

4. Appendix

4.1. A: Initial supported tables

Table	Supported since xRM release
ADDRESS	2022.2.0
ACTIVITY	2022.2.0
ACTIVITYLINK	2022.2.0
COMMUNICATION	2022.2.0
CONTACT	2022.2.0
ORGANISATION	2022.2.0
PERSON	2022.2.0

4.2. B: ADITO specific faker functions

To deal with some adito specific cases like keywords or attributes we implemented our own faker module called **adito** with its own specific functions

4.2.1. adito

Module to deal with ADITO specific cases.

4.2.1.1. adito.keyword

Pick a random keyid of a adito keyword.

4.2.1.1.1. Parameters

Name	Type	Default	Description
pContainerName	string		the container name of a adito keyword to load a random keyid

Returns: string

Example

```
faker.adito.keyword(pContainerName: string): string
faker.adito.keyword("OrganisationType") // "ORGPARTNER"
```

4.2.1.2. adito.pickFromDb

Pick a random value of a specific table and field from the database.
The data will be selected once and cached during execution.

Make sure not to select to much data cause of the caching.

4.2.1.2.1. Parameters

Name	Type	Default	Description
pTableColumn	string		"table.column" where to select the data
[pCondition]	string		sql condition to reduce the amount of data to select

Returns: string

Example

```
faker.adito.pickFromDb(pTableColumn: string, [pCondition]: string): string
faker.adito.pickFromDb("SALUTATION.SALUTATION", "SALUTATION.ISOLANGUAGE = 'deu'") // "Herr" -> picks a salutation from the database
faker.adito.pickFromDb("CONTACT.CONTACTID") // "441c7327-6857-4cbd-9159-3f9577651156" -> picks a contactid
```

4.3. C: Set up AditoFaker without the preset of xRM 2022.2.0

If you need to install AditoFaker manually please do the following steps:

- Make sure you have done all steps from 2.1.1 Setup before installation.
- Add the adito-faker dependency to your package.json
- Generate the folder structure as mentioned in the initial configuration

- Get the current TableDefinition.json from the xRM and place it into the folder
- [optional] Get the current configuraiton.json from the xRM and place it into the folder
- Get the current scripts from the xRM and place it under ./tools/AditoFaker/scripts/ as mentioned in the designer scripts
- Add the scripts to your package.json (see the current xRM or the screenshot)

You can copy the faker dependency here, the necessary scripts here and the tools folder with the scripts here
Make sure to check if a new faker version is available and update the dependency if necessary.

Now you should be good to go.
Continue with 2.1.2 Install AditoFaker.

4.4. D: example configurations, best practices, usage of faker functions

Coming soon.