Data Validation - Code Generation

10 - June - 2025 v 0.1

Agenda

- Details about the code generation for Data Validation
  - Codebook Schema
  - Codebook Schema JSON
  - Codebook
  - How to use the code generator
  - How to use the code generated by code generator

Codebook Schema

- Codebook is a JSON File
- Codebook Schema
  - Metadata about the Codebook JSON
  - Codebook_schema.json follows the schema standards - https://json-schema.org/specification
  -

CodeBook Schema - High level

No	Name	Type	Mandatory	Comments
1	version	string	yes	Version of the Codebook Set to “1.0.0”
2	questions	Array of objections of Question Type	yes	Contains information about the questions

CodeBook Schema - Question Type

No	Name	Type	Mandatory	Comments
1	qid	integer	yes	Question number should be greater than 0
2	versionAdded	string	yes	In which codebook version this was added
3	versionDeleted	string	no	In which codebook version this was deleted
4	type	string	yes	One of the following - "int", "string", "bool", "datetime", "timestamp"
5	isMandatory	bool	yes	Ensure that the value of the field is not NULL in the db
7	preprocRules	Array	no	Contains the pre processing to be done on the value to transform the value
8	validationRules	Array	no	Contains the validations to be done on the value
9	conditionalRules	Array	no	Contains the validations to be done on the value in

CodeBook Schema - Pre-Processing Type

No	Name	Type	Mandatory	Comments
1	name	string	yes	Name of the pre-processing function will be based on this name
2	params	array	no	Array of the parameters, in addition to the value that is checked, to be passed to above function.

CodeBook Schema - Validation Rules Type

No	Name	Type	Mandatory	Comments
1	name	string	yes	Name of the rule function will be based on this name
2	params	array	no	Array of the parameters, in addition to the value that is checked, to be passed to above function.

CodeBook Schema - Conditional Rules Type

No	Name	Type	Mandatory	Comments
1	name	string	yes	Name of the function will be based on this name
2	params	array	no	Array of the parameters, in addition to the value that is checked, to be passed to above function.
3	qids	array	yes	The question numbers whose answer has to be used in the comparison along with the answer to the given question

Codebook Schema JSON


```
{
  "type": "object",
  "properties": {
    "version": {
      "description": "The version of the Codebook.json schema.",
      "type": "string",
      "pattern": "^\\d+\\.\\d+\\.\\d+$"
    },
    "questions": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "qid" : {
            "description": "The question id. Should match with the entry in the Database table.",
            "type": "number",
```

Code Generator

- One Python File - generate.py
- Expects three parameters
  - The code book schema json file
  - The code book json file
  - The output file
- Dependent on jsonschema library
  - pip install jsonschema
- `python3 generate.py -s codebook_schema.json -c codebook.json -o generated code.py`
- `python3 generate.py –schema=codebook schema.json –codebook=codebook.json –output=generated_code.py`

Code Generator - Only Mandatory fields and One Question. (isMandatory is False)

Codebook JSON

```
{
    "version": "1.0.0",
    "questions": [
      {
        "qid": 1,
        "versionAdded": "1.0.0",
        "isMandatory": false,
        "type": "int",
        "name": "Header",
        "description": "ODK Metadata Header"
      }
    ]
}
```

Generated Code - Import List

```
from common_utils import *

from preproc_rules import *

from validation_rules import *

from conditional_rules import *
```

- The functions called by the generated code is expected to be manually coded in these files.

Generated Code - Validation Functions

```
####

# Name : Header

# Description : ODK Metadata Header

####

def validate_qid_1(val) :

  try :

    val = val.strip()

    new_val =  convert_dt(val, 'int')

    return (new_val, "")

  except Exception as e:

    return (val, str(e))
```

- One validate function per qid
  - validate_qid_<qid number>()
- strip() and convert_dt() called for all.
- All answers are stored in DB as string.
  - Convert_dt converts string to correct data type
- Return values are tuples of value and error string
  - Empty string indicates succes.
    - Use the new_val returned
  - A non empty string indicates convert_dt or pre-processing or validation failed.

Generated Code - Conditional Rules

```
 # This function will handle all conditional checks for all
questions

 # Input  - dictionary of items. question id is the key and
the answer is the value

 #

 # Output - Tuple (None, None) indicates success

 #        - Tuple (qid, error_string) indicates failure

 #

 ###

 def validate_conditional_rules(vals):

   return (None, None)
```

- `validate_conditional_rules` function to be called to check validations when a questions answer is dependent on answer to other question(s)

Code Generator - Only Mandatory fields and One Question. (isMandatory is True)

Codebook JSON

```
{
    "version": "1.0.0",
    "questions": [
      {
        "qid": 1,
        "versionAdded": "1.0.0",
        "isMandatory": true,
        "type": "int",
        "name": "Header",
        "description": "ODK Metadata Header"
      }
    ]
}
```

Generated Code - Validation Functions

```
def validate_qid_1(val) :

  if val == None:

    return (val, 'Mandatory field value cannot be None')



  try :

    val = val.strip()

    new_val = convert_dt(val, 'int')



    return (new_val, "")



  except Exception as e:

    return (val, str(e))
```

- If the field is mandatory, the validation will fail if the value is None

Code Generator - Optional preprocRules

Codebook JSON

```
{
    "version": "1.0.0",
    "questions": [
      {
        "qid": 1,
        …
        "description": "ODK Metadata Header"
        "preprocRules" : [{
          "name": "multiply_by_if_less_than",
          "params": [20, 100]
        }
        ]
      }
    ]
}
```

- The example shows one Pre Processing Rule
- Can have any number of pre processing rules
- “Params” are optional
- The pre processing functions might change the value

Generated Code - Validation Functions

```
def validate_qid_1(val) :

  if val == None:

    return (val, 'Mandatory field value cannot be None')



  try :

    val = val.strip()

    new_val = convert_dt(val, 'int')



    # Call the pre processing rules

    new_val = preproc_rule_multiply_by_if_less_than(new_val, 20, 100)
```

- The pre processing function that needs to be manually implemented is
  - `preproc_rule_multiply_by_if_less_than(new_val, 20, 100)`
- “preproc_rule” is prefix
- “multiply_by_if_less_than” is picked from the “name” field of codebook
- “20, 100” are picked from the “params”
  - If no “params” are specified, the function will have only “new_val” argument

Manual Code - preproc_rules.py

```
def
preproc_rule_multiply_by_if_less
_than(val, multiplier, compare)
:

    if val < compare :

        return val * multiplier



    return val
```

- Always return (modified or original) value
- Not expected to change data type
- Raise exception if there is any error
- Code generator provides hints
  - `python3 generate.py -s codebook_schema.json -c 1.json -o 1.py`

`The following pre processing functions need to be implemented :`

`preproc_rule_multiply_by_if_less_than(new_val, 20, 100)`

Code Generator - Optional validationRules

Codebook JSON

```
{
    "version": "1.0.0",
    "questions": [
      {
        "qid": 1,
        …
        "description": "ODK Metadata Header"
        "validationRules" : [{
          "name": "limit_range",
          "params": [1, 50]
        }
        ]
      ]
      }
    ]
```

- The example shows one Validation Rule
- Can have any number of validation
- “Params” are optional
- The validations functions are not expected to change the value

Generated Code - Validation Functions

```
def validate_qid_1(val) :

  if val == None:

    return (val, 'Mandatory field value cannot be None')



  try :

    val = val.strip()

    new_val = convert_dt(val, 'int')



    # Call the validation rules

    validate_rule_limit_range(new_val, 1, 50)
```

- The pre processing function that needs to be manually implemented is
  - `validate_rule_multiply_by_if_less_than(new_val, 20, 100)`
- “validate_rule” is prefix
- “limit_range” is picked from the “name” field of codebook
- “1, 50” are picked from the “params”
  - If no “params” are specified, the function will have only “new_val” argument

Manual Code - validation_rules.py

```
def validate_rule_limit_range(new_val,
min, max) :

     if new_val < min :

          raise Exception(f"value {new_val}
is less than supported minimum of :
{min}")



     if new_val > max :

          raise Exception(f"value {new_val}
is more than supported maximum of :
{max}")
```

- Raise exception on error
- Not expected to change data type
- Code generator provides hints
  - `python3 generate.py -s codebook_schema.json -c 1.json -o 1.py`

```
The following validate functions need to be
implemented :



validate_rule_limit_range(new_val, 1, 50)
```

Code Generator - Optional conditionalRules

Codebook JSON

```
{
    "version": "1.0.0",
    "questions": [
      {
        "qid": 1,
        …
        "description": "ODK Metadata Header"
        "conditionalRules" : [{
          "name": "not_greater_than",
          "qids": [2]
        }
      ]
      }
    ]
}
```

- The example shows one Conditional Rule
- Can have any number of conditional rules
- “Params” are optional
- The conditional functions are not expected to change the value

Generated Code - Validation Functions

```
def validate_conditional_rules(vals):

  try :


conditional_rule_not_greater_than (vals[1], vals[2])

  except Exception as e:

    return (1, str(e))



  return (None, None)
```

- The conditional function that needs to be manually implemented is
  - `conditional_rule_not_greater_than(vals[1], vals[2])`
- “conditional_rule” is prefix
- “not_greater_than” is picked from the “name” field of codebook
- The first arg is value of this question. The next argument(s) are the answer(s) to the question(s) specified in “qids” field
- If “params” field is present, those are passed as arguments

Manual Code - conditional_rules.py

```
def
conditional_rule_not_greater_than(vals_1,
vals_2):

    if vals_2 > vals_1 :

        raise Exception("invalid")

    return
```

- Raise exception on error
- Not expected to change data type
- Code generator provides hints
  - `python3 generate.py -s codebook_schema.json -c 1.json -o 1.py`

The following Conditional Rules functions need to be implemented :

`conditional_rule_not_greater_than(vals[1], vals[2])`

Manual Code - common_utils.py

```
def convert_dt(val, dt) :

    match dt:

        case "string" :

            return val

        case "int" :

            return int(val)

        case _ :

            raise Exception(f"Unknown data
type : '{dt}'' for conversion")
```

- Val will always be null
- Dt will be a string which will be set to the “type” specified in codebook
- Return value of proper data type
- Raise exception on error

How to use the generated file

Things to take care

- Implement all required functions in
  - Common_utils.py
  - Preproc_rules.py
  - validation_rules.py
  - Conditional_rules.py
- The generated python code imports the above files
  - You may manually change the python to remove the import of above file(s) and import your file(s) which has implementation of the required functions.

How to call the validation function

- Option 1
  - (new_val, error) = validate_qid_1(val)
    - This will call the validation and pre-process rules for question id 1
- Option 2
  - fn = globals()[f"validate_qid_{qid}"]
  - (new_val, error) = fn(val)
    - This can be used to call functions dynamically

How to handle return value validation function

- Return value is a tuple of value and error string
- If error string is not empty, it means the validation has failed.
- If the error string is empty, it means validation is successful.
- Please note that the value might have been modified by the validation function.
  - Compare the string representation of the returned value with the original value (which is always string) to identify if value was changed or not.

How to call the conditional function

- Create a dictionary with key as question id and the answer to that question as its value.
- The dictionary should contain ALL question id's that are mentioned in ALL the "conditionalRules" specified in the codebook.json
- (qid, err) = validate_conditional_rules(vals)
  - If err is empty string there was no error.
  - Else there was an error (and qid tells which question id's conditional rule returned error)
- If there are lots of conditional functions, let us modify the code generator to generate one function per question (similar to validation and pre processing rules.)