# **Workflow Engineer Interview / Practical part**

**Conditions:**

You are given a REST API server that provides information about kids’ toys and games in which those toys were used.

host: kids.example.com

schema: http

Endpoints:

Endpoint 1. '/toys' - returns list of all toys in the following YAML format:

**toys**:

**- id**: 1

**name**: boat

**status**: broken

**status\_updated**: 2018-03-19

**games**:

**- id**: 1

**note**: need repair

**- id**: 14

**note**: boat is broken

**- id**: 7

**name**: Teddy Bear

**status**: ok

**status\_updated**: 2018-03-30

**games**:

**- id**: 5

**note**: bear feels well

**- id**: 43

**name**: octopus

**status**: ok

**status\_updated**: 2018-03-19

**games**:

**- id**: 5

**note**: felt rather good though had no water to swim

**- id**: 14

**note**: two tentacles are lost

**Notes:**

* + - status - the field may take the following values: "ok", "broken", "repair"
    - status\_updated - date of last status update

Available GET parameters (all are optional and may be used in one request):

* + - regex\_lang - language to use regular expression format (e.g.: "python", "php" - use language you know better)
    - note\_regex - regular expression in format according to "regex\_lang" rule. If set, a result containing only toys for which `games.note` value matches given regular expression.
    - updated\_after (format: YYYYMMDD) - return all entries where status\_updated > updated\_after
    - updated\_before (format: YYYYMMDD) - return all entries where status\_updated < updated\_before

Endpoint 2. '/games' - list of all games in YAML format:

**games**:

**- id**: 1

**name**: Ships in the ocean

**date**: 2018-02-12

**- id**: 5

**name**: ZOO Railroad

**date**: 2018-03-30

**- id**: 14

**name**: Octopus-destroyer

**date**: 2018-03-18

Available GET parameters (all are optional and may be used in one request):

* + date\_from (format: YYYYMMDD) - return all entries with date >= date\_from
  + date\_to (format: YYYYMMDD) - return all entries with date <= date\_to

Data on API server is renewed every day, but requests should be done **once a week**.

Imagine we have a simple script that can make HTTP requests and make SQL INSERT requests with results according to provided YAML configuration. Configuration format is the following (example data are set):

**conf\_id**: runner # unique identifier - string (digits and lowercase Latin letters)

**request**:

# URL of endpoint

**url**: [http://example.com/users?date={CURRENT\_DATE('%Y-%m-%d')}&prev\_date={DATE\_WITH\_OFFSET('%Y-%m-%d](http://example.com/users?date=%7bCURRENT_DATE('%25Y-%25m-%25d')%7d&prev_date=%7bDATE_WITH_OFFSET('%25Y-%25m-%25d)')}

**date\_offset**: # may be omitted

**days**: 1

**response**: # configure processing of HTTP response

**- id**: users # unique identifier - string (digits and lowercase Latin letters)

**path**: users # path to entry containing data list

**sql\_table**: users # name of SQL table to insert data

**fields**:

**- origin**: id

**sql\_field**: user\_id

**- origin**: name

**sql\_field**: user\_name

**- origin**: age

**sql\_field**: user\_age

**- id**: apps

**path**: users.applications # if dot is used, a `parent.child` node is assumed

**sql\_table**: apps # name of SQL table to insert data

**fields**:

**- origin**: id

**sql\_field**: app\_id

**- origin**: date

**sql\_field**: app\_date

**- origin**: parent.id # if dot is used in `fields.origin` entry, one-level parent is assumed

**sql\_field**: parent\_id

**Notes:**

**1**. url may contain the following placeholders in curly braces:

* + CURRENT\_DATE(format) - prints in date of script execution in specified format (use one from programming language you prefer, say, for python CURRENT\_DATE('%Y-%m-%d') will print "2020-01-01" when run January 1, 2020).
  + DATE\_WITH\_OFFSET(format) - prints in date calculated as above CURRENT\_DATE + value specified in `date\_offset`. Only `days` and `months` offsets are allowed. Values may be negative. For example, for

**date\_offset**:

**months**: 1

and python date formatting style DATE\_WITH\_OFFSET('%Y-%m-%d') will print "2020-02-01" when run on January 1, 2020.

**2**. It is assumed, that the following MySQL request will be made with each result row:

**INSERT** **INTO** `sql\_table`**(**sql\_field1**,** sql\_field2**,** **...)** **VALUES** **(**value1**,** value2**,** **...);**

where `sql\_table` is table name specified in config entry, sql\_field1,2,.. - is field name specified as `sql\_field`. Values will be taken from API response.

For example, with above configuration the script may consume the following HTTP API response:

**users**:

**- id**: 1

**name**: Jimmy

**age**: 18

**applications**:

**- id**: 567

**date**: 2019-05-12

**- id**: 3345

**date**: 2019-06-13

and will make the following SQL requests:

**INSERT** **INTO** `users`**(**`user\_id`**,** `user\_name`**,** `user\_age`**)** **VALUES** **(**"1"**,** "Jimmy"**,** "18"**);**

**INSERT** **INTO** `apps`**(**`app\_id`**,** `app\_date`**,** `parent\_id`**)** **VALUES** **(**"567"**,** "2019-05-12"**,** "1"**);**

**INSERT** **INTO** `apps`**(**`app\_id`**,** `app\_date`**,** `parent\_id`**)** **VALUES** **(**"3345"**,** "2019-06-13"**,** "1"**);**

## **Given the above information, complete the following tasks:**

**Notes:** your answers should be saved in the files that are listed in the `Where to answer` column of the table below. In the case of sql requests, you should save them in one file, but each request should have a comment, for example:

**SELECT** **\*** **FROM** `users`**; # Task 1**

**SELECT** **\*** **FROM** `groups`**; # Task 2**

**...**

|  |  |  |
| --- | --- | --- |
| # | Task | Where to answer  (create the file first) |
| 1 | Write down MySQL requests to create `toys`, `games` and `toys\_games` tables (`toys\_games` is a relation table to connect `toys` and `games`) containing all information from above "toys" API (host kids.example.com).  toys:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | id | toy\_id | name | status | status\_updated |   games:   |  |  |  |  | | --- | --- | --- | --- | | id | game\_id | name | date |   toys\_games:   |  |  |  |  | | --- | --- | --- | --- | | id | game\_id | toy\_id | note | | Test.sql |
| 2 | Write down MySQL request to create `toys\_repair` table to contain information on toys being repaired. Table should contain fields `id` (MySQL internal primary key), `toy\_id` (toy `id`), `issue\_description` (here you will find the `note` field values for each `toy\_id` field (one `toy\_id` contains many `note`)).  toys\_repair:   |  |  |  | | --- | --- | --- | | id | toy\_id | issue\_description | | Test.sql |
| 3 | Create configuration for the script to retrieve the following: *see points* ***a-c*** *below*.  **Notes:**   * Data should not overlap with data received earlier. * Field configuration should be appropriate for inserting data into tables from pp.1 and 2. | |
| 3.a | All available data about games for the last week (current date included). Should fill in table from **Task 1**. | a.yaml |
| 3.b | All available data about toys for the last week (current date included). Should fill in tables from **Task 1**. | b.yaml |
| 3.c | All data about toys containing "repair" or "break" or "broken" substring in `note` field. Should fill in table from **Task 2**. | c.yaml |
| 4 | Create MySQL request to get all available data about toys and their status updates during last year: `toy\_id`, `toys.name`, `status`, `status\_updated`, `games.name`, `date`, `note`. | Test.sql |
| 5 | Create MySQL request to get list of toys never been repaired. | Test.sql |