FUNCTIONAL AND TECHNICAL DOCUMENT

TECHOPS-399: UPLOADED OTRS DOCUMENTS

TEAM AUTOMATION

TECH-OPS

PURPOSE:

The purpose of this functional and technical document is to provide documentation on the requirements of Automating the document uploading process in OTRS system. This script, what it does is, it can find the tickets under the respective articles in OTRS which has the valuable attachments and it can track these ticket details by having them locally and it will switch the queue of the tickets which has an relevant attachment after applying required conditions.

SCOPE:

The tickets which has attachments (attached by customer) can be tracked at any time separately in a convenient manner. It can be moved to the relevant queue “Contact::Documents” to reduce the confusions and make this process consistent.

We are using “pandas” data frames to do all the dml operations in the DB. It has a variety of utilities to perform Input/Output operations in a seamless manner.

We are using “pyotrs” module to do update operation in the otrs core tables. This is the rest API, which can do this process properly. So we can track the changes on the ticket clearly and also it can avoid to making the DB connection over and over again.

Tables and schema used:

- otrs
  - article
  - ticket
- otrscase
  - Doc_Upload
  - Ticket_list

Technical functionality:

In terms of technical functionality, We have created a separate function for each step as explained below.

config:

We have an config for OTRS scripts, which contains the OTRS credentials such as (username,password,server,database
URL) in a key:value pattern of python dictionary.

It can be accessed by calling the key For create a DB connection as, mysql+pymysql://' + “username" + ‘:’+ "password" + '@' + "server" + ':3306/' + "database"

For make a pyotrs connection,("URL", "username”+ ‘@’+ “server", "password").

Function-1 (initial_insert):

This function can fetch all the data, which are satisfied with the below filter conditions by joining otrs.article ,otrs.ticket and otrscase.Ticket_List table. Also eventually it can make an entry (insert) in otrscase.Doc_Upload table.

The table Doc_Upload has been created for this process and in that table, we will store all the results we have got by running this script.

Filter conditions:

- Value in the column ‘Id’ of article table should be greater than the maximum number present in the column ‘article_id’ of otrscase.Doc_Upload table. This condition can ignore the data which we have done this process already.
- Customer ID should not be null in otrs.ticket table.
- Queue ID should be 5. (integer ‘5’ represents the contact queue).
- Article sender type ID should be 3. (integer ‘3’ represents the sender type as customer).
- Exclude the articles, which we have received from the below domains.
    1. MicrosoftExchange
    2. Personifyfinancial
    3. Woopra
    4. Postmaster
    5. Localhost

Below are the Fields, which will get inserted by this function,

1. article_id
2. ticket_id
3. ticket number
4. Email
5. Title
6. content_path
7. create_time

8. contact_id
9. LoanID

We will be using ‘to_sql’ function from pandas module to make this insert in an efficient way.

Function-2 (Attachment_path):

- This function can create a attachment path for the respective article, for which the Attachment_Path is null and doc_uploaded is false in the otrscase.Doc_Upload table. Below are the steps,

- Fetch the article_id, Email and content_path from otrscase.Doc_Upload table where the Attachment_Path is null and doc_uploaded =’0’.

    1. article_id = incremental interger.
    2. Email = Email address of the customer, for whom the documents belongs to.
    3. content_path = current date in YYYY/MM/DD format.

- Create a variable with the following pattern of path /opt/otrs/var/article/+’content_path’+’article_id’.

- Update the derived path from the above step in the otrscase.Doc_Upload table as Attachment_Path.

Function-3 (Find_attachment):

This function is used to identify, if the article have any documents attached with the following extension (".jpeg",".jpg",".gif",".png",".pdf") and if it is present then it can make an entry in otrscase.Doc_Upload table as ‘True’. The steps are below,

- Fetch article_id, ticket_id, ticket number, Attachment_Path, contact_id from DB by joining article and Doc_Upload table and apply the below filter conditions,

- In Doc_Upload table the field “Attachment_Path” in is not null and have_attachment is not equal to '1'.

- In article table, “a_subject” field is not equal to 'Ticket Merged'.

- We can create a variable as valid_attachments and assign the following valuable file extensions (".jpeg",".jpg",".gif",".png",".pdf").

- By using the for loop take the article's one by one from the result of the first step.
- create a variable as path and assign the path, which we retrieved from DB.
- Create an empty list as 'l'.
- By Using the os module we can open the file path and have that as another variable 'f'.
- With the help of another for-loop we can check the files inside the path, if any files found inside the path with the extensions mentioned above, then it can append that inside the empty list 'l'.
- Now we can update the have_attachment field in 'otrscase.Doc_Upload' such as True or False based on the list 'l'.
- IF the list 'l', which we get from the above function- is not empty then it can be updated as '1' else '0'.

Function-4 (update ticket):

- In this function we have to update the queue id as '25' in queue table and Uploaded_Attachments as '1' in Doc_Upload table for those who we have attached the 'have attachment ' field as '1' in the above function-3. At first the following fields (ticket_id, ticket number, article_id , have_attachment) can be fetched from the Doc_Upload table where it has have_attachment as '1' and Uploaded_Attachments as null.
- By using a for loop in the query result then we have to check if the ticket comes under Contact::Fax or Contact::Mail Documents. If yes then we can exclude the ticketID else,
- we can pass the ticketID in the update call which we have made by using the pyotrs API and we can update the queue_id as '25' (25 represents the queue "Contact::Documents").
- For this update we will be using python wrapper module PyOTRS.
- In this module we can simply import the client function and we can create a session by calling client.session_restore_or_create() function. It uses a temporary file on the hard drive to store the Session ID (just like cookies in a browser) and avoids sending the

username+password (and therefore creating a new session) on every API call.

PFB, the sample code to make connection,

```
try:

    client = Client("url", "useername@localhost", "password")
    client.session_restore_or_create()
    client.ticket_get_by_id(1)


except Exception as e:
    print(e)
```

In the ticket_get_by_id() function, we can pass the ticket_id directly, so that we will connect with it and we can do the required operations on the particular ticket_id.

PFB, the sample code for Update the field.

```
    my_article = ticket({"queue_id": "25"})
    client.ticket_update(ticketID, ticket=my_article)
```

As mentioned in the snip we can pass the required field as a python dictionary and we can make this process properly.

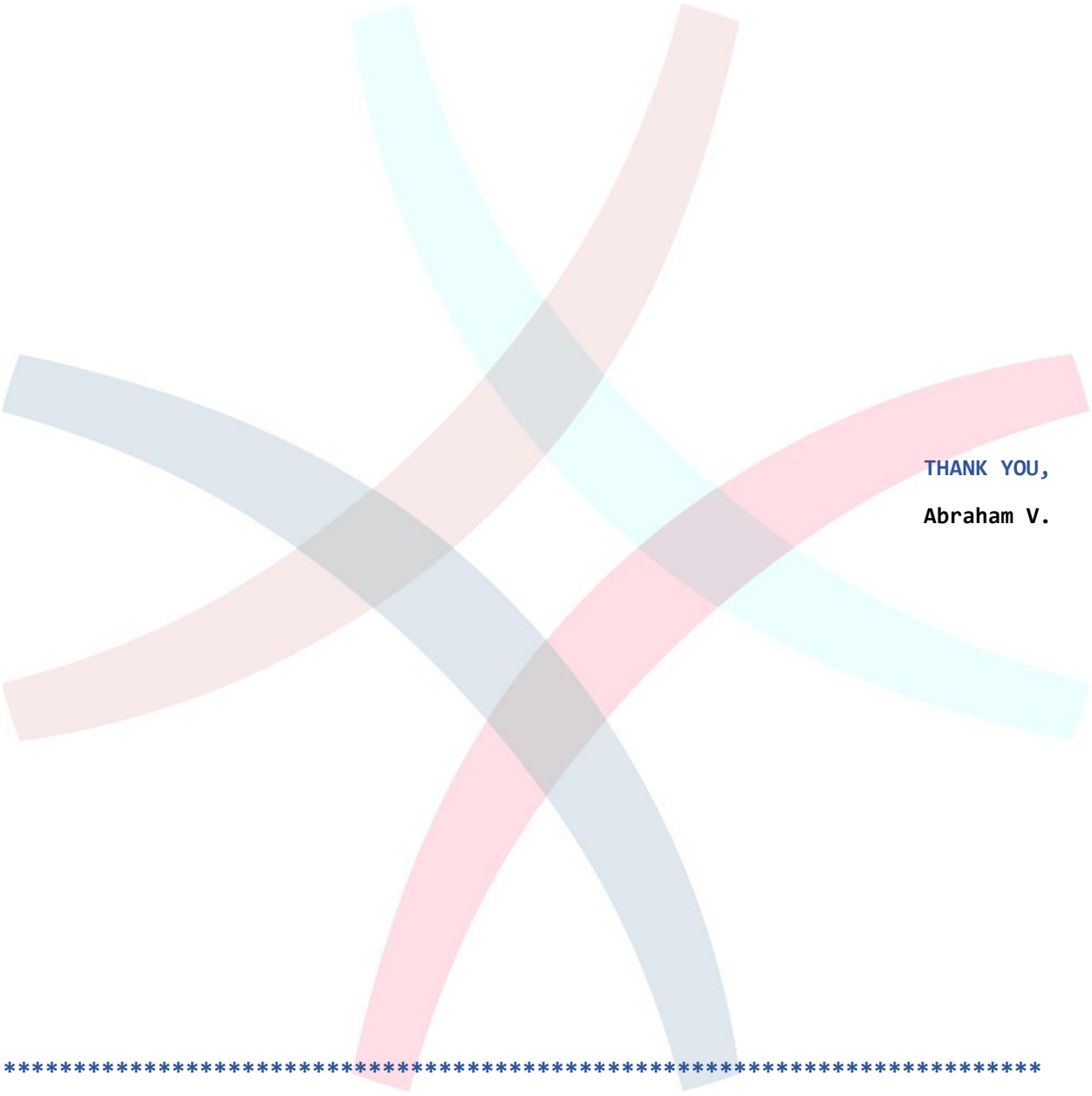
At-last this function can update the Uploaded_Attachments field as ‘1’ in otrs.Doc_Upload table for which tickets we have update the queue_id as ‘25’.

Exception and null handling:

- In the initial function, if there is no data present, while querying with the required conditions then at that point the script will get terminated and the email message will be sent out to the recipients.
- try and exception block should be used in each function of this script to capture the exceptions. So, if any data couldn’t make successful progess as expected then the email can be send out as an exception

email by using our send mail functionality.

- It will trigger an acknowledgement email at the end of the process, which contains ticketID’s which not had an attachment.



THANK YOU,

Abraham V.

****