Data Engineer Interview Exercises

**Read before you start:**

* The programming language is limited to Python/SQL. If you used any open-source packages, please include the package name and the way to install them.
* Try to lay out your thoughts in a concise fashion.

1. Based on the given table with 3 rows of example data:
   * Please use SQL to display the gap (in days) between transactions for each customer.
   * Please use SQL to “detect” the customers with unusually large gaps. You can use Snowflake specific functions. (**extra points**)

-- create table customer

-- (

-- cust\_id int,

-- trans\_date date,

-- profit int

-- )

-- select \* from customer

-- insert into customer values

-- (001,'2020-10-09',45),

-- (001,'2021-01-13',98),

-- (002,'2017-08-12',456);

/\* Gap(in days) between transaction for each customer partitioned by cutomers \*/

-- select cust\_id, trans\_date,

-- trans\_date - LAG(trans\_date) OVER (PARTITION BY cust\_id ORDER BY trans\_date)

-- AS gap\_in\_days from customer

/\* Gap(in days) between transaction \*/

-- select

-- cust\_id,

-- trans\_date,

-- trans\_date - lag(trans\_date) over (order by trans\_date) as gap\_in\_days

-- from customer

/\* with unsually large gaps(in days) between transactions \*/

-- with diff\_table as

-- (

-- select

-- cust\_id,

-- trans\_date,

-- trans\_date - lag(trans\_date) over (order by trans\_date) as gap\_in\_days

-- from customer

-- )

-- select \* from diff\_table

-- where gap\_in\_days > 500 /\*lets assume treshold vlaue is 500\*/

In snowflake we can use **DATEDIFF** function to find difference in dates.

|  |  |  |
| --- | --- | --- |
| cust\_id | trans\_date | profit |
| 001 | 2020-10-09 | 45 |
| 001 | 2021-03-13 | 98 |
| 002 | 2017-08-12 | 456 |

1. Write **your own** function to determine if the input string is a valid numeric value. For instance, [“0”, “ 0.1 “, “1e10”, “ 2E-5”, “-10e3”] are valid; while [“abc”, “5e”, “-+2”, “e7”] are not.
   * Besides the provided cases, please come up with at least 5 more meaningful test cases. Implement and verify all test cases. **please do not use float () function from Python**.

def string\_function(string):

try:

int(string)

return True

except ValueError:

try:

complex(string)

return True

except ValueError:

return False

# // to find with the input string

str = input("enter the string:")

if string\_function(str):

print("valid")

else:

print("not valid")

# provided test cases.

test\_cases = ["0","0.1 ","1e10","2E-5","-10e3","abc","5e","+-2","e7",]

# other test cases.

test\_cases = ["12.54.1","-3.14","3-2j","-32e","10."]

for case in test\_cases:

result = string\_function(case)

print(f"{case}: {result}")

1. Use the attached data (p3\_data.tar.gz). Please:

* Produce a summary of the data.

Here, in my opinion the data resembles the patient feedback given to the hospital application. ‘1’ indicates the positive feedback and ‘0’ indicates negative feedback.

* Use Python to ingest the data into a Snowflake database.

import snowflake.connector

import pandas as pd

# Snowflake connection parameters

snowflake\_params = {

'user': '\*\*\*\*\*\*\*\*\*\*',

'password': '\*\*\*\*\*\*\*\*\*\*\*\*',

'account': '\*\*\*\*\*\*\*\*\*\*\*\*',

'warehouse': 'COMPUTE\_WH',

'database': 'AMERIGAS',

'schema': 'PUBLIC'

}

# CSV file path

csv\_path = '/Users/sidharthaannamaneni/Downloads/p3\_data.csv'

# Snowflake table name

table\_name = 'FEEDBACK'

# Create a Snowflake connection

conn = snowflake.connector.connect(\*\*snowflake\_params)

# Create a Snowflake cursor

cur = conn.cursor()

print("connected to snowflake")

try:

stage\_name = 'temp\_stage'

create\_stage = f"CREATE TEMPORARY STAGE {stage\_name}"

cur.execute(create\_stage)

put\_sql = f"PUT [file://{csv\_path}](file:///%7bcsv_path%7d) @{stage\_name}"

cur.execute(put\_sql)

print("file pushed to stage from local")

copy\_cmd = f"COPY INTO {table\_name} FROM @{stage\_name}/p3\_data.csv FILE\_FORMAT=(TYPE=CSV, FIELD\_DELIMITER=',', SKIP\_HEADER=1)"

cur.execute(copy\_cmd)

print("file written to table")

conn.commit()

except snowflake.connector.errors.DatabaseError as e:

print(f"Error {e}")

finally:

cur.close()

conn.close()

* Assuming the data size is 20x larger or more, how would you change your implementation? (**extra points**)

The given data is of 20mb approx. It took few seconds for me to ingest into database. So 20 times of the given data could take just few minutes.

Also snowflake supports bulk loading. I can process the ingestion in parallel so we can load large files also effectively.

1. Customers leave. Sometimes a customer would come back due to the following reasons. Such as: 1) we offer the best deal in the area; 2) we made some effort to win this customer back; 3) our competitors no longer provide this area. When this happens, the customer’s status will be changed from “left” to “active” and use the same ID. Based on the above context: (**extra points**)

* Describe your approach to identify these “regained” customers. (Assume we have data for everything)

Approach1:

Here, assuming we have data for everything. We can use the transaction data of the customers. When find the usual gaps in the transactions of any customer. If there is any gap between the transaction(months/years). Then we can trace out the customers from the transaction table with their Cust\_ID.

Approach2:

Assuming the customer table available with all the details. Any customer when he is registered with the company. The created\_date is stored and updated\_date is stored. Using these fields, we can list out the regained customers. If the created\_date and updated\_date are same, then the customer comes under regular customer. If the updated\_date is different from created\_date then that means updated\_date is altered/updated. Here we compare these created\_date field and updated\_date field and list out the regained customers.

1. Please explain what data version control is and write your thoughts on how to achieve that (**extra points**)

Data Version Control:

DVC is a tool and approach for tracking and managing changes to data and machine learning models. It is designed to handle the challenges of data science project like versioning and collaboration. It is designed to function on top of git repositories and cloud storage. DVC stores large files and datasets outside of Git in independent storage. These files can either be stored locally on the user's PC or on a big cloud storage service. We can also setup remote repository on any server and connect to it remotely.

Manage the data and organize the folder structure, naming conventions, and file formats for easy tracking, finding changes and updated files. We can use suitable version control systems as the foundation for version control. Here we can use GIT which is widely used and well-integrated with DVC. User can collaborate and track changes like branching and merging effectively. Integrating with Git helps to create a unified workflow for managing both code and data. Additionally, DVC provide functionality specifically designed for managing data in machine learning projects, model files and experiment tracking. Along with source code and data files also saves important metadata such as data collection methods processing steps and descriptions. Using DVC for collaboration and clear communication channels to ensure everyone understand the status of different data versions, updates, and new findings among team members. Lastly, we can conduct audits and reviews to ensure the effectiveness of the project goal. By following these guidelines, we can achieve strong data version control in any projects.