Data – Technical Test

Data Analysis

We are seeing some large fluctuations in the main KPI of our website —something we call the "bad to good ratio"— that is, the ratio of users who have "quality" to users that don't have it. We want to find the reasons for these fluctuations.
After some investigation, we think that the underlying cause can be related to changes in the traffic sources. We have requested our development team to extract some data, and we are attaching a sample of it:

- **bad.csv: CSV file containing the ones with no quality. An example of a row would be 104474, 3e14b0c730317aad79307c3c214fc838, "2017-09-21 09:23:43", where 3e14b0c730317aad79307c3c214fc838 is the identifier of the user, 104474 is the identifier of the source of the traffic and "2017-09-21 09:23:43" is the date of the event.**
- **good.csv: CVS file containing the same information for quality ones. In order to analyze this data, the business team would like several graphs and info:**

1. **One graph representing how many no quality users we need in order to generate a quality one, for all the users of the website. For example, a value of 45 would indicate that for each 45 bad quality users, one is of good quality, on average. Ideally, we would like to have a point each 5 minutes and each point would be an average ratio of the last hour events.**

The solution is on the attached "1.bad_to_good.py" file

2. **Generate the same graph for the top 10 sources of traffic, so we can see the ratio for the biggest sources of traffic.**

The solution is on the attached "2.bad_to_good_top10.py" file

3. **In addition to the graphs, the business team would really appreciate some conclusions about the possible influence of the traffic source on the ratio, so analyze them and add any valuable info. Any statistical analysis to try to find correlations between the source of the traffic and the main ratio will be highly valued.**

In the attached "3.ratio_by_source.py" file we've singled out the lowest performing sources. We could observe that 29 of the 111 sources had a bad_to_good ratio of over 50, 9 of them being over

100. Taking those 29 out of the equation though, didn't impact the average bad to good ratio the way we thought, as we can see the graph not changing much and the average lowering very lightly.

4. **Finally, we would like to consider the implementation of some Business Intelligence tool where we could easily monitor this data and detect problems in the future. What would you suggest?**

There are many BI tools we could use to visualize and monitor time series such as the ones on this exercise. A good one could be Power BI, a powerful tool that would allow us to visualize real time updates on the bad to good ratio status and detect fluctuations in quality in real time.

Data Architecture

We are implementing an e-commerce website from scratch. On this website we will have multiple articles from multiple manufacturers. The most important user interactions create events that we want to store, process and visualize.
What user events would you collect and why? How would you organize the data architecture to achieve this and what technologies would you use or implement?

The main user events we should collect would be:

- <u>Product views</u>: whenever a user views a product an event containing the user id, product id, product name, category and timestamp is generated.
- <u>Product click</u>: whenever a user clicks on a product or goes to the product's page an event containing the user id, product id, product name, category and timestamp is generated.
- <u>Search</u>: whenever a user searches for something we should collect the searched text, the number of results and the timestamp.
- <u>Add to cart</u>: adding products to the cart should also be stored as an event with the product id, name, number of items, price, timestamp and userid.
- <u>Wishlist</u>: adding a product to the wishlist should also generate an event containing the product id, name, number of items, price, timestamp and userid.
- <u>Buys</u>: whenever there's a sale an event containing the userid, order id, product ids, product names, number of items and individual and total prices.

Other useful events depending on the website's features would be:

- <u>Wishlist</u>: whenever a user adds a product to their wishlist an event containing the user id, product id, product name, category and timestamp is generated.
- <u>Product rating</u>: whenever a user rates a product an event containing the user id, product id, product name, rating and timestamp should be created.
- <u>Share</u>: if the website had a sharing feature an event containing the user id, the product id, the product name, the sharing method and the timestamp should be created.

- <u>User login</u>: whenever a user logs in an event containing the user id, username, location, and timestamp should be generated.

The fields contained in these events are surface level and there could be more if we wanted to go more in depth. With the these events we could extract conclusions and calculate statistics on products and users. Things like trends, most desirable products, badly performing products, user buying profiles, location based behaviors, products being bought together, and many other things.

In terms of data architecture, our data should be stored on a data warehouse, since it would be our own structured data and it would use a defined schema, so a data lake wouldn't be necessary. We could use a Hadoop Ecosystem to ingest and model our data through key-value.

SQL Performance

Are the following Transact-SQL snippets good or bad practices from a performance perspective?

- **Searching for all rows of the year 2012:**

CREATE INDEX tbl_idx ON tbl (date_column);
SELECT text, date_column
FROM tbl
WHERE datepart(yyyy, date_column) = 2012;

Creating an index on the column in which we are going to select from will speed up the query's execution, but the selection itself being done with the datepart function could slow it down since it has to compute the year from the date_column for every row. A better solution for the selection of the date would be to use a range, such as:
WHERE date_column >= '2012-01-01' AND date_column < '2013-01-01'

- **To find the most recent row:**

CREATE INDEX tbl_idx ON tbl (a, date_column);
SELECT TOP 1 id, date_column
FROM tbl
WHERE a = @a
ORDER BY date_column DESC

This snippet is good practice from a performance perspective. The index created on both "a" and "date_column" will help speed up the selection (WHERE a = @a) and sorting (ORDER BY date_column

DESC) of the data. Also, using "TOP 1" together with "ORDER BY date_column DESC" makes sure that the query will only return the most recent row and it will avoid the scanning of the entire table.

- **Two queries, searching by a common column:**

CREATE INDEX tbl_idx ON tbl (a, b);

SELECT id, a, b

FROM tbl

WHERE a = @a

AND b = @b;

SELECT id, a, b

FROM tbl

WHERE b = @b;

This third snippet creates an index on both "a" and "b" which can help speed up the first query's execution. However, the second query's search is only on "b", making the index ineffective in this case. A better solution would be to create two separate indexes, one on "a" and one on "b," so that each query could benefit from them (the first query from both indexes and the second from the one on "b").

SQL Quality

Comment the problems of the attached Transact-SQL stored procedure (sql_quality.sql) and provide a better solution.

I am not familiarized enough with table and procedure creation to give an informed answer to this question. I would need time to get informed and to work on the matter to be comfortable giving an answer.