D205 – Data Acquisition Performance Assessment

Bernard James Connelly III

Master of Science, Data Analytics, Western Governors University

Dr. Sewell

October 18, 2024

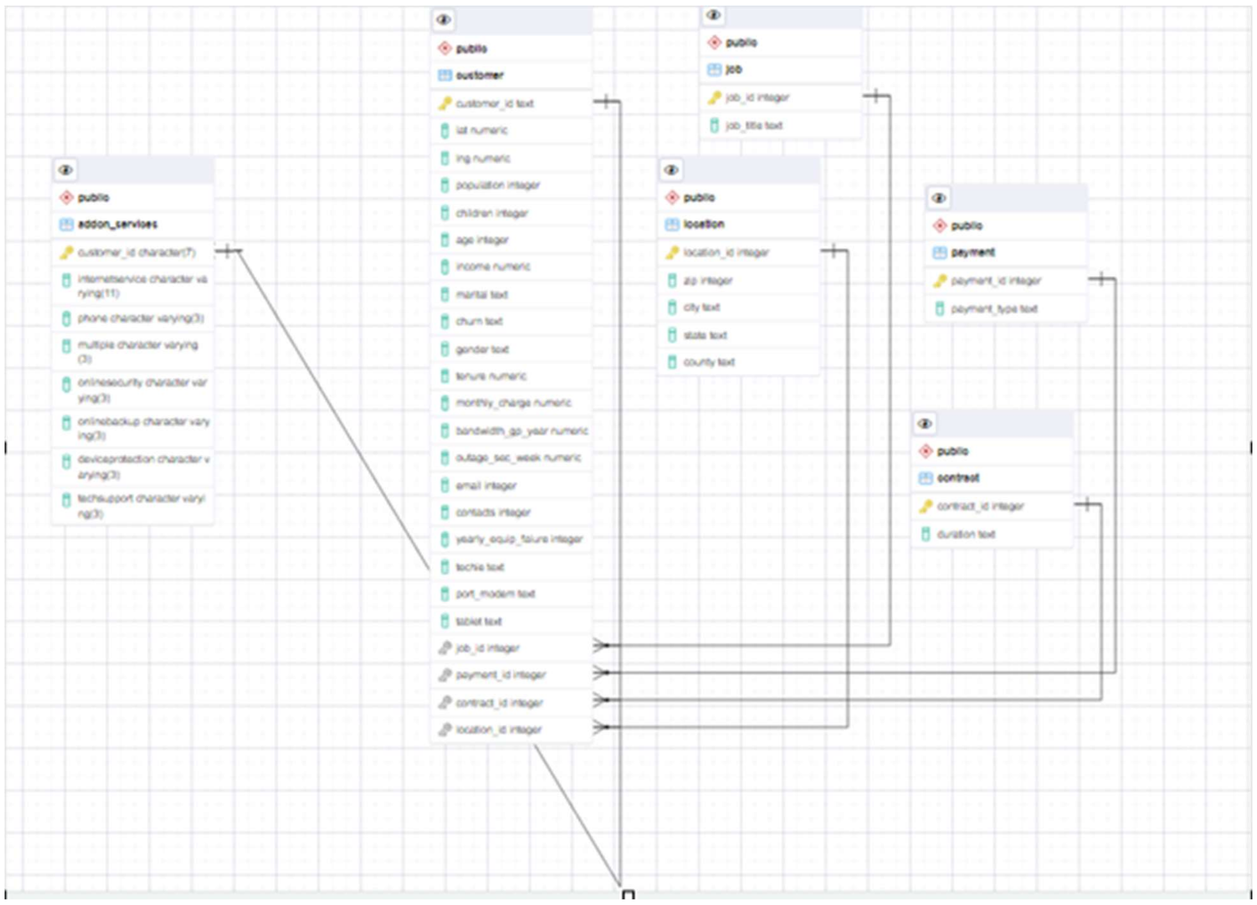
A. Research Question

The research question I chose for this project is "Are customers who self-identify as tech-savvy more likely to purchase add-on services - specifically related to Online Security?" This is a relevant question to the business as they could cater discounts or bundled packages offered to technologically capable customers. The company could identify and generalize these findings across all locations to increase overall revenue and engagement.

A1 + A2: What data to utilize and how to answer the research question

To answer this question, I will create a table housing the relevant columns for analysis. Data from the *customers* table from the central database will be joined with the *services* add-on table with the shared field customer ID as a key to join the data. The *customers* table will utilize the *techie* column to identify those who self-identify as technologically capable. The add-in CSV file *services.csv* will utilize the *OnlineSecurity* column to identify customers who purchased the additional internet protection. These values will be pulled from the database into the results file to identify customers who identify as techies and whether they determined that they need the *OnlineSecurity* service. The *customer_id* column will be imported as char(7), since it is a unique identifier always containing seven characters. The *techie* and *onlinesecurity* columns will be input as varchar(3) columns as both hold either "yes" or "no" responses.

B. ERD



The above Entity Relationship Diagram was generated using the GUI in PostgreSQL. The primary tables utilized are the *customer* and *addon_services*, however, the GUI created a diagram for all tables in the database. The relationship between *customer* and *addon_services* is represented as a 1:1 relationship, as there can be one and only one response in the *addon_services* table for each unique *customer_id* from the customer table. Additionally, the default settings for relational constraints exist between the two tables. As *customer_id* is the primary key, the not null relational constraint exists, meaning the primary key cannot have any null values. As a foreign key, *customer_id* ensures referential integrity as each value must match an existing value in the referenced table's primary key and enforces an "on delete no action" constraint, which prevents deleting a referenced row in the parent table so long as the child table has dependent rows.

<u>B. 1: Table Relationships and ERD Issues</u>

The existing *customer* table and *addon_services* table have a 1 to 1 relationship. This means there is one row for each unique customer ID in each table instead of several, which may be expected in a table with a one-to-many relationship. The primary shortcoming in the diagram is that pgAdmin does not support creating one-to-one relationships in an ERD Diagram. This can be noted in the documentation for pgAdmin, where the table relationships only identify 1M (one-to-many) and MM (many-to-many) with no reference to creating a one-to-one relationship. (PostgreSQL pgAdmin 4.8.13 documentation, n.d.). The remainder of the relationships developed in the database inherently have these relationships built-in, but the *customers* table to *addon_services* table relationship was modified to reflect the appropriate relationship.

<u>B. 2: SQL Statement to create *addon_services* table</u>

```
-- CREATION OF ADD-ON CSV TABLE
CREATE TABLE public.addon_services
(
        customer_id char (7) NOT NULL,
        internetservice varchar(11),
        phone varchar(3),
        multiple varchar(3),
        onlinesecurity varchar(3),
        onlinebackup varchar(3),
        deviceprotection varchar(3),
        techsupport varchar(3),
        PRIMARY KEY (customer_id),
        CONSTRAINT customer_id_fkey FOREIGN KEY (customer_id)
```

```
                REFERENCES customer(customer_id) MATCH SIMPLE
                ON UPDATE NO ACTION
                ON DELETE NO ACTION
);
```

The above SQL statement was created using pgAdmin's query function. All fields were imported to the table from the CSV file, despite only *customer_id and onlinesecurity* being utilized in the research question. The table values were preserved by using char (7) for *customer_id* as this was a fixed value identifier of 7 characters, and varchar was utilized to preserve the data in the original file. Additionally, customer_id was made as both the primary and foreign key for the tables to establish the necessary relationships and relevant constraints to ensure referential integrity.

B. 3: SQL Statement to load data from CSV file

```
--Importing data from the CSV file into the addon_services table
COPY addon_services
FROM 'C:\LabFiles\Services.csv'
DELIMITER ','
CSV HEADER;
```

The above SQL statement was created in pgAdmin, where the CSV file was loaded from the virtual desktop and put into pgAdmin to manipulate via PostgreSQL. The copy line item identifies the table for data to be loaded into, and it identifies the location of the CSV file in the virtual desktop. The delimiter field exists to specify a comma is the delimiting value in the file. Finally, the CSV Header line tells the program to ignore the first row in the file, as it is a header with column names.

C. SQL Statement to respond to the research question

```
--Query to show customers who self-identify as techies and opt-in/out for online security
```

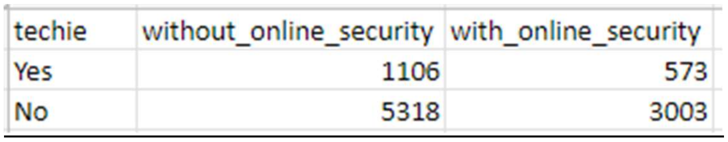
```
SELECT
    c.techie,
    COUNT(CASE WHEN a.onlinesecurity = 'No' THEN 1 END) AS without_online_security,
    COUNT(CASE WHEN a.onlinesecurity = 'Yes' THEN 1 END) AS with_online_security
    FROM customer c
            INNER JOIN addon_services a ON c.customer_id = a.customer_id
            GROUP BY c.techie
            ORDER BY c.techie desc;
```

The above query was generated to solve the research question in part A of this document. For this assignment, the above query sets out to retrieve a count of the number of customers who opted into online security and did not, then grouped it according to customers who self-identified as technologically capable. The first portion of the query pulls in the *techie* field from the *customer table*. The subsequent two lines count the number of unique rows where the responses in the *onlinesecurity* column of the *addon_services* table listed “no” and “yes” respectively occur using a CASE WHEN clause. These are each given their own AS statements to make the resultant datafile's header clearer and cleaner. The two tables were joined on the *customer_id* field in each table as this unique identifier would occur once for each row in the data. The inner join portion of the join cause is implicit since there is only one row per customer id. Group by ensures the resultant set provides one row each for customers who self-identify as techies and those who do not. The final order by line puts the responses in descending order, as reading the yes responses on top of the no presents a more natural look to the results.

<u>C.</u> **<u>1: Resultant Data File</u>**

D205_Results_File_B
ernard_Connelly.csv

The above CSV file was the resultant dataset from the query, exported using pgAdmin's GUI. A screenshot of the resultant set is visible below. At a glance at the data, the total value of the numbers therein represents 10000 records, indicating the full range of customers available was used based on the data dictionary.



techie	without_online_security	with_online_security
Yes	1106	573
No	5318	3003

<u>D.</u> **<u>+ D1: Refresh Frequency and justification</u>**

Based on the churn database dataset's data dictionary, customers can update or terminate their contracts every month. Due to this fact, refreshing the data in the file on a monthly basis makes the most sense to capture the most relevant updates for the business. Any changes to the population intra-month would either be additions of new customers or modifications to individual plans, which would not significantly change the rates of techies to online security users. By refreshing monthly, the company can provide a consistent timeframe of analysis to see changes month-over-month. It can also pull in additional fields that may be related, such as other device protection, tech support, or churn statistics, to draw different conclusions from the data. These analyses can then be actioned via grouping packages of services, or offering additional discounts at the end of registering customers based upon their self-identified responses housed in the customers table.

<u>E.</u> **<u>Panopto Recording – Uploaded in PA UI</u>**

F. Citation

pgAdmin Development Team. (n.d). PostgreSQL pgAdmin 4.8.13 documentation. Retrieved from -

https://www.pgadmin.org/docs/pgadmin4/development/erd_tool.html#table-relationship-options