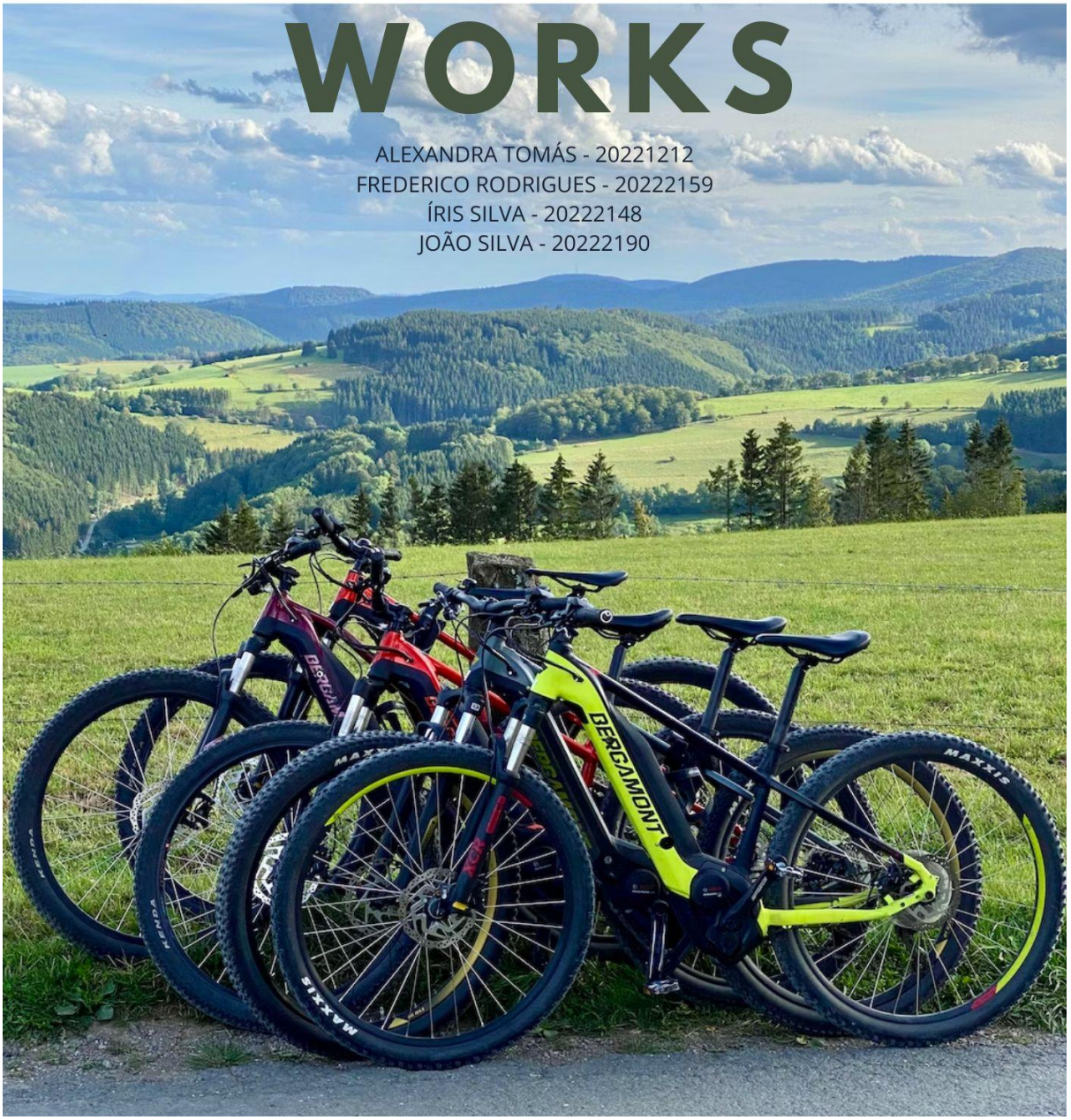
SQL PROJECT - GROUP O

ADVENTURE WORKS

ALEXANDRA TOMÁS - 20221212
FREDERICO RODRIGUES - 20222159
ÍRIS SILVA - 20222148
JOÃO SILVA - 20222190



Introduction

Adventure Works is a company involved in the manufacturing of bicycles, as well as the production of some of the key components required for their assembly, in addition to accessories and apparel. Adventure Works has an online store where they sell their products directly to the final customer, as well as to retail stores and wholesalers. Every year before the announcement of new bicycle models, there is still a considerable stock of old models, preventing the successful launch of new bicycle models.

The company needed to address two issues. Firstly, to set up an online auction covering all products for which a new model was expected to come out in two weeks' time, to outflow the old stock as much as possible. To do so, it was necessary to extend the current database model to support all auction specifications, therefore, a new schema Auction was created to house this new database extension. Secondly, to determine the optimal locations for their first two physical stores in the United States, taking into consideration that they do not want to compete directly with the stores that buy their products for resale.

Business Problems

Stock Clearance

The first stored procedure, uspAddProductToAuction, adds a product as auctioned. It takes in three parameters: the product ID, the ExpireDate, and the InitialBidPrice. First, it checks if the product has already been auctioned. If not, it checks if the product can be auctioned based on sell end date and discontinued date. It then checks if the original price of the product is zero, which means it cannot be auctioned. If the expiry date is not specified, it sets it to one week from the current date. If the initial bid price is not specified, it calculates it based on the list price of the product and a flag value. Finally, it checks if the initial bid price exceeds the maximum possible value. After validating the input, the stored procedure inserts the product into the Auction table with the specified initial bid price and expiry date. It also inserts a record into the BidHistory table to record the auction event. If the validation fails or the insertion fails, an error is raised, and the transaction is rolled back.

The second stored procedure, uspTryBidProduct, is responsible for adding bids on behalf of customers. It takes in three parameters: the product ID, the customer ID, and the bid amount (which defaults to NULL). The procedure first declares various parameters that will be used throughout its execution. It retrieves the highest bid made so far (or the auctions initial value), the minimum increase bid and the maximum bid percentage from the BidSettings table. It also gets the original price and name of the product from the Product table. Next, it checks if the customer ID exists and throws an error if it does not exist. It then checks if the product is currently being auctioned and throws an error if it is not. If the bid amount is not specified, it calculates the bid amount based on the highest bid price and the minimum increase bid. It then checks if the bid amount exceeds the maximum bid limit and throws an error if it does. Finally, it checks if the bid amount is lower than the highest bid made so far and throws an error if it is. If all the checks are passed, the procedure inserts the bid into the Bid table

and updates the BidPrice in the Auction table. It also inserts the bid history into the BidHistory table, which includes the product ID, customer ID, bid amount, and timestamp. The procedure is wrapped in a TRY-CATCH block to handle any errors that may occur during execution. If an error occurs, it rolls back the transaction and raises a user-friendly error message.

The third stored procedure, uspRemoveProductFromAuction, is designed to remove a product from being listed as auctioned, even if there might have been bids for that product. The stored procedure first checks whether the product exists in the Auction. If the product is not found in the Auction table, the stored procedure raises an error, however, if the product is found in the Auction table, the stored procedure begins a transaction. It then deletes all bids associated with the product from the Bid table and it also deletes the product from the Auction table. After deleting the product from both tables, the stored procedure inserts a history record into the BidHistory table, indicating that the product has been removed from auctions. The history record is created using a variable called History that concatenates a string with the product name retrieved from the Production. Product table and a message indicating that the product has been removed from auctions. If any errors occur during the execution of the stored procedure, the CATCH block is executed, and it then raises a user-friendly error message. Finally, the stored procedure rolls back the transaction.

The fourth stored procedure, uspListBidsOffersHistory, is designed to retrieve bid history data for a specified customer within a given time interval. The procedure takes four input parameters: the customer ID, the start time, the end time, and a flag indicating whether to retrieve data for active auctions only. The procedure first checks that the start time is prior to the end time, raising an error if this is not the case. Then selects bid history data from the BidHistory table, left joining the Product table to include product information. Filtering then the data by the specified customer ID and the given time interval, the Active flag is used to determine whether to retrieve history for products that are currently auctioned (Active = 1) or to retrieve all bid history regardless of whether the product is currently auctioned or not (Active = 0). Finally, it will return the bid history and creation date for the specified customer and time interval, ordered by creation date in descending order.

The last stored procedure, uspUpdateProductAuctionStatus, performs two actions on the Auction table: it updates the auction status for completed and canceled auctions, and it logs a bid history for completed auctions. For completed auctions, the stored procedure inserts a new record into the BidHistory table, recording the product, the winning customer, and the time of the auction end. It then updates the Auction table, setting the auction status to "Completed" (status ID of 2) and updating the bid price and the highest bidding customer for the auction. For canceled auctions, the stored procedure inserts a new record into the BidHistory table, recording the product and a message indicating that the auction has been canceled due to lack of bids. It then updates the Auction table, setting the auction status to "Canceled" (status ID of 3). If an error occurs during the transaction, the stored procedure rolls back the changes made and raises an error message with details about the error.

Brick and Mortar Stores

The second business challenge Adventure Works faced was identifying the best locations to open their first two Brick and Mortar stores in the United States. The company's goal is to sell directly to customers while avoiding direct competition with stores that buy their products for resale, this way maintaining a strong relationship with their reselling partners. To tackle this issue, a data-driven approach was used to examine multiple factors, including existing customer data, purchase value, geographic distribution, and individual sales categorized by city.

First, the cities with the best sales were identified. The query was divided into 2 common table expressions (CTEs). The TopStoreCustomerSales CTE selects the top 30 store customers in the US and their corresponding cities, groups them by CustomerID and city, calculates the sum of total sales for each group and orders the results by total sales in descending order. Next, the RemainingCitiesIndividualSales CTE selects individual customers in the US, filters the results to exclude the cities from the TopStoreCustomerSales CTE, groups them by city and state, and calculates the sum of total sales for each group. The final SELECT statement in the query retrieves the city, state, and total sales from the RemainingCitiesIndividualSales CTE, and orders the resulting data by total sales in descending order to prioritize the cities with the highest individual Customer sales. The T-SQL code for this step and its results can be found in the annex (Fig. 1 and Table 1).

Afterwards, the top potential cities for the first two stores were identified. A separate query was formulated, which was divided into 3 common table expressions (CTEs). The first CTE is the TopStoreCustomerSales, already used and explained in the previous query. The second CTE, ClientInfo, selects individual customers and their information like Total Purchase year to date (YTD) and Yearly income from the Person.Person table, and filters the results to include only customers identified as Individuals. The third CTE, ClientValue, joins the ClientInfo CTE with address and state information, filters the results to include only US cities, groups the results by city and state, and calculates the number of customers, sum of total purchase YTD, average total purchase YTD and average yearly income for each group. The results are then filtered to include only groups with more than 10 customers.

The final SELECT statement in the query retrieves the city, state, number of customers, sum and average of total purchase value, and average yearly income, and orders the resulting data by the number of customers in descending order. In this SELECT, cities included in the TopStoreCustomerSales CTE were excluded.

The T-SQL code for this step and its results can be found in the annex (Fig. 2 and Table 2).

By employing a data-driven approach, an analysis was conducted on individual customer data, encompassing the number of customers, total and average purchase value, and average yearly income in various cities throughout the United States. The study concluded that the most favorable areas for launching new stores are Berkeley, California, and Burien, Washington.

Berkeley, California, was chosen due to its strong individual buyer sales of 73,730.01 and a large number of customers. Berkeley also boasts a high average total purchase year-to-date (YTD) and a substantial average yearly income for its customers. These factors indicate a strong market

potential and a higher likelihood of success for an Adventure Works store in the area. Establishing a store in Berkeley presents an excellent opportunity for Adventure Works to capitalize on this market by catering to the demands of customers in the area, who not only have a proven interest in Adventure Work products but also possess the financial means to support the business.

Burien, Washington, was selected based on its significant number of customers. Opening a store in Burien will not only increase AdventureWorks' market presence in Washington but also ensure a more extensive geographic distribution of stores. By expanding to Burien, it is possible to diversify store presence across multiple states, rather than concentrating all stores in California.

Both cities, Berkeley and Burien, have demonstrated strong current values, such as Purchase Year-to-Date and Yearly Income. Additionally, they appear in our Top Sales per City rankings, indicating that they are excellent investment opportunities. These cities have consistently performed well in the past and continue to show great potential in the present, making them ideal locations for our new stores.

By setting up stores in both Berkeley and Burien, Adventure Works will effectively cater to the demands of customers in these regions, capitalize on the existing market potential for their products, and expand their market presence in the United States. Furthermore, the choice to diversify store locations across different states promotes a balanced distribution and helps mitigate the risk of market saturation in any single region.

Limitations

During the development of this work, limitations were identified within the second business problem, corresponding to opening Brick and Mortar Stores. In fact, only data regarding the current customers was analyzed. Gathering data concerning potential customers in the area under consideration could be beneficial to provide a better understanding of the market potential to open a new store.

Conclusions

Regarding the first business problem, extending the Adventure Works database to support product auctions, five tables were created to support the required five stored procedures. As a result, it is now possible for Adventure Works to set up a product to be auctioned and enable customers to create bids on their items. Removal of products from being auctioned is also allowed, and the auctions status can be changed accordingly (either to be auctioned, completed or canceled). Additionally, all the changes made to auctions and bids are recorded in the BidHistory table. The stored procedures were also made so that any possible mistakes in the stored procedures inputs would be handled.

As for the second business problem, it is recommended to open in Berkeley and Burien in order to avoid cannibalization while capitalizing on existing market potential that was witnessed based on income and total purchases of Adventure Works customers. Additionally, the decision to diversify

store locations across different states promotes a balanced distribution and mitigates the risk of market saturation in any single region.

Annexes

```sql
-- List of Store Customers in the US
With TopStoreCustomerSales AS (
    SELECT top 30
        c.CustomerID,
        a.city,
        SUM(soh.TotalDue) as TotalSales
    FROM Sales.SalesOrderHeader soh
    JOIN Sales.Customer c ON soh.CustomerID = c.CustomerID
    JOIN Sales.Store s ON c.StoreID = s.BusinessEntityID
    JOIN Person.BusinessEntityAddress bea ON s.BusinessEntityID = bea.BusinessEntityID
    JOIN Person.Address a ON bea.AddressID = a.AddressID
    JOIN Person.StateProvince sp ON a.StateProvinceID = sp.StateProvinceID
    WHERE c.StoreID IS NOT NULL AND sp.CountryRegionCode = 'US'
    GROUP BY c.CustomerID, a.city
    Order by TotalSales DESC
), -- New query with only Indivitual Buyers in the US, excluding the cities from the top 30 Stores
RemainingCitiesIndividualSales AS (
    SELECT
        a.City,
        sp.Name AS State,
        SUM(soh.TotalDue) as TotalSales
    FROM Sales.SalesOrderHeader soh
    JOIN Sales.Customer c ON soh.CustomerID = c.CustomerID
    JOIN Person.BusinessEntityAddress bea ON c.PersonID = bea.BusinessEntityID
    JOIN Person.Address a ON bea.AddressID = a.AddressID
    JOIN Person.StateProvince sp ON a.StateProvinceID = sp.StateProvinceID
    WHERE a.City NOT IN (SELECT City FROM TopStoreCustomerSales)
    AND c.StoreID IS NULL
    AND sp.CountryRegionCode = 'US'
    GROUP BY a.City, sp.Name
) -- Top cities, counting only indidivual buyers
SELECT TOP 10
    city,
    State,
    TotalSales
FROM RemainingCitiesIndividualSales
ORDER BY TotalSales DESC;
```

Fig.1 T-SQL code for the cities with the highest individual total sales, excluding the cities from our Top 30 stores.

```sql
WITH TopStoreCustomerSales AS (
    SELECT top 30
        c.CustomerID,
        a.city,
        SUM(soh.TotalDue) as TotalSales
    FROM Sales.SalesOrderHeader soh
    JOIN Sales.Customer c ON soh.CustomerID = c.CustomerID
    JOIN Sales.Store s ON c.StoreID = s.BusinessEntityID
    JOIN Person.BusinessEntityAddress bea ON s.BusinessEntityID = bea.BusinessEntityID
    JOIN Person.Address a ON bea.AddressID = a.AddressID
    JOIN Person.StateProvince sp ON a.StateProvinceID = sp.StateProvinceID
    WHERE c.StoreID IS NOT NULL AND sp.CountryRegionCode = 'US'
    GROUP BY c.CustomerID, a.city
    ORDER BY TotalSales DESC
),
ClientInfo AS (
    SELECT
        BusinessEntityID,
        Demographics.value('declare namespace aw="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";
        (/aw:IndividualSurvey/aw:TotalPurchaseYTD)[1]', 'decimal(10,2)') as TotalPurchaseYTD,
        Demographics.value('declare namespace aw="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";
        (/aw:IndividualSurvey/aw:YearlyIncome)[1]', 'varchar(10)') as YearlyIncome
    FROM Person.Person
    Where PersonType = 'IN'
),
ClientValue AS (
    SELECT
        COUNT(ci.BusinessEntityID) AS NumCustomers,
        SUM(ci.TotalPurchaseYTD) AS SumTotalPurchaseYTD,
        AVG(ci.TotalPurchaseYTD) AS AvgTotalPurchaseYTD,
        AVG(CAST(
            CASE
                WHEN CHARINDEX('-', ci.YearlyIncome) > 0
                THEN
                    (CAST(SUBSTRING(ci.YearlyIncome, 1, CHARINDEX('-', ci.YearlyIncome) - 1) AS decimal) +
                    CAST(SUBSTRING(ci.YearlyIncome, CHARINDEX('-', ci.YearlyIncome) + 1, LEN(ci.YearlyIncome)) AS decimal)) / 2
                ELSE NULL
            END AS decimal)) AS AvgYearlyIncome,
        a.City,
        sp.Name AS State
    FROM
        ClientInfo AS ci
    JOIN Person.BusinessEntityAddress bea ON ci.BusinessEntityID = bea.BusinessEntityID
    JOIN Person.Address a ON bea.AddressID = a.AddressID
    JOIN Person.StateProvince sp ON a.StateProvinceID = sp.StateProvinceID
    WHERE sp.CountryRegionCode = 'US'
    GROUP BY a.City, sp.Name
    HAVING COUNT(ci.BusinessEntityID) > 10
)
SELECT
    NumCustomers,
    SumTotalPurchaseYTD,
    AvgTotalPurchaseYTD,
    AVGYearlyIncome,
    City,
    State
from ClientValue
Where City NOT IN (SELECT city FROM TopStoreCustomerSales)
Order by NumCustomers DESC
```

Fig. 2 T-SQL code that retrieves by city and state the total number of customers, the total sum of their purchases, the average of purchase by customer and their yearly income, for US customers.

	city	State	TotalSales
1	Bellflower	California	334018,0869
2	Burbank	California	305487,1492
3	Berkeley	California	285243,0051
4	Colma	California	258528,0009
5	Chula Vista	California	257925,0512
6	Burien	Washington	254547,2501
7	Bremerton	Washington	250925,2273
8	Burlingame	California	249358,325
9	Concord	California	232542,6845
10	Beverly Hills	California	223980,1525

Table 1. Results obtained from T-SQL stated in Figure 1. Table represents the city and state with the highest individual total sales, excluding the cities from the Top 30 stores

	NumCustomers	SumTotalPurchaseYTD	AvgTotalPurchaseYTD	AVGYearlyIncome	City	State
1	212	25321.49	119.440990	23691.438095	Burien	Washington
2	212	44096.12	208.000566	24946.630434	Concord	California
3	210	16488.73	78.517761	24387.439613	Beaverton	Oregon
4	206	54066.83	262.460339	24755.232954	Chula Vista	California
5	200	73730.01	368.650050	25096.153409	Berkeley	California
6	198	43660.08	220.505454	24847.351955	Burlingame	California
7	194	41751.29	215.212835	24625.254335	Bellflower	California
8	192	28576.97	148.838385	23956.762195	Burbank	California
9	188	23323.36	124.060425	24341.466257	Beverly Hills	California
10	182	27013.58	148.426263	24799.325966	Bremerton	Washington

Table 2. Results obtained from T-SQL stated in Figure 2. Table represents the city and state with the highest number of customers as well as the number of customers, total purchases, average purchase by customer and yearly income.