Advanced Database Manipulation

Prepared by: North Star Consultants (GROUP 1)

Date: November 12, 2023

Submitted by:

Yian Chen

Sean Adam Kagugube

Emmanuella Acheampong

Susan Arzapalo

Introduction

North Star Consultants

North Star Consultants is a leading consultancy dedicated to designing and implementing cutting-edge Database Management Systems (DBMS) for businesses across various industries. With a team of highly skilled database experts and a passion for data optimization, we specialize in helping financial institutions harness the full potential of their data resources.

Mission Statement:

"Our mission is to offer comprehensive DBMS solutions that enable financial institutions to unlock the true value of their data assets and ensure that our clients thrive in a rapidly evolving financial landscape."

Vision Statement:

"To be the trusted partner of choice for financial institutions seeking excellence in optimizing data storage, retrieval, and analysis, ensuring our clients stay ahead in today's data-driven world".

Our Services:

- **Data Quality Assurance**: We ensure your data is accurate, consistent, and reliable, eliminating errors and inaccuracies.
- **Data Transformation**: We convert raw data into actionable insights, seamlessly integrating data from multiple sources.
- **Insights for Success**: We provide client-centric insights to empower strategic planning and drive success.
- **Engaging Reporting**: Our visualizations and reports offer clarity and actionable

recommendations.

Why Invest in us?

We have a history of over 100+ successful database implementations and satisfied clients in our 2 years of existence. Our solutions are high in demand and our expertise and commitment to excellence ensure investor confidence in our ability to execute and deliver results.

Customer Background

Horizon Bank is a prominent bank with over 50 branches across the United States. The bank wants to launch a new loan product for its loyal customers (more than 2 years with the bank). In addition, the bank is grappling with soaring marketing expenditures and an alarming drain on resources due to inefficient marketing campaigns. Horizon Bank is in dire need of a database with insights for optimizing their marketing strategies targeting the right customers, reducing expenses, and maximizing the effectiveness of their campaigns to maintain a competitive edge in the financial sector.

Value Proposition

Phase_1: To Build a relational database for managing the data set of the bank and leverage advanced data analytics and predictive modeling

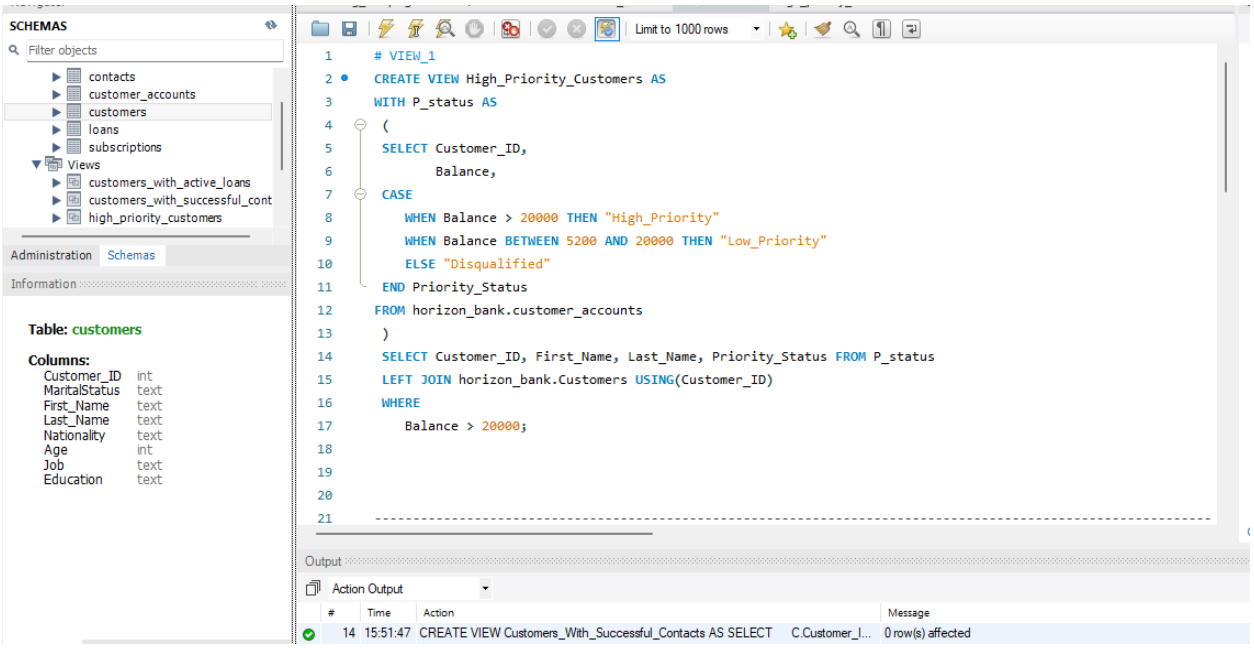
Phase_2: To identify customers who qualify for the loan loyalty product for targeted marketing

Phase_3: To provide customer insights to enable the bank to develop a marketing strategy that minimizes expenses while delivering superior results.

1. VIEW_1: High_Priority_Customers

Description:

The High_Priority_Customers view is designed to identify and categorize customers with balances above $20,000, classifying them as "High_Priority." This view aligns with the bank's objective of targeting top-priority customers for its new term deposit product that offers an attractive interest rate for customers with high balances.



The view’s benefit to Horizon Bank:

Branch Manager Engagement: The view is crucial for executive management, especially the branch manager, as it identifies top-priority customers. These high-value customers will be personally contacted by the branch manager, ensuring a more personalized and effective marketing approach.

Access for End Users:

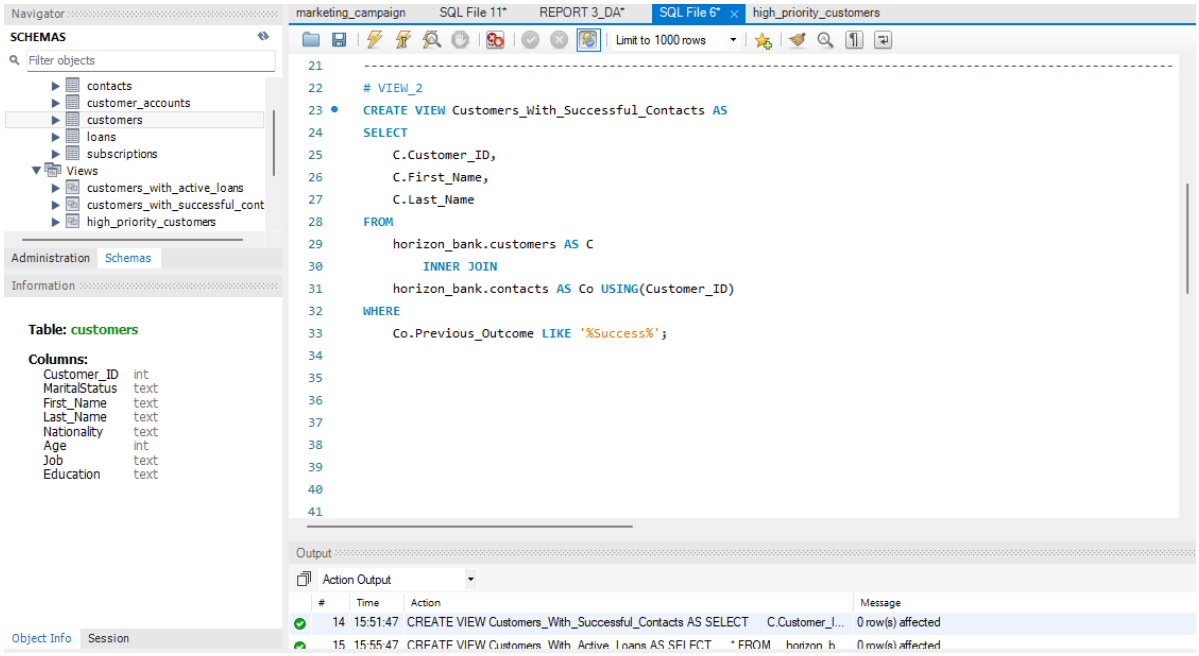
The report generated from this view will be shared with executive management, including the

branch manager. To accommodate the non-tech-savvy nature of these users, North Star Consultants will ensure that the report is accessible through user-friendly interfaces or even directly delivered as periodic reports.

VIEW_2: Customers_With_Successful_Contacts

Description:

The Customers_With_Successful_Contacts view identifies customers who have had positive interactions with the bank, offering insights for improved customer relationship management and marketing efficiency.



The view's benefit to Horizon Bank:

Enhanced Customer Engagement: This view empowers the contact center team to engage consistently with customers who have a history of successful interactions. It improves loyalty and enables the bank to foster stronger relationships with its customer base.

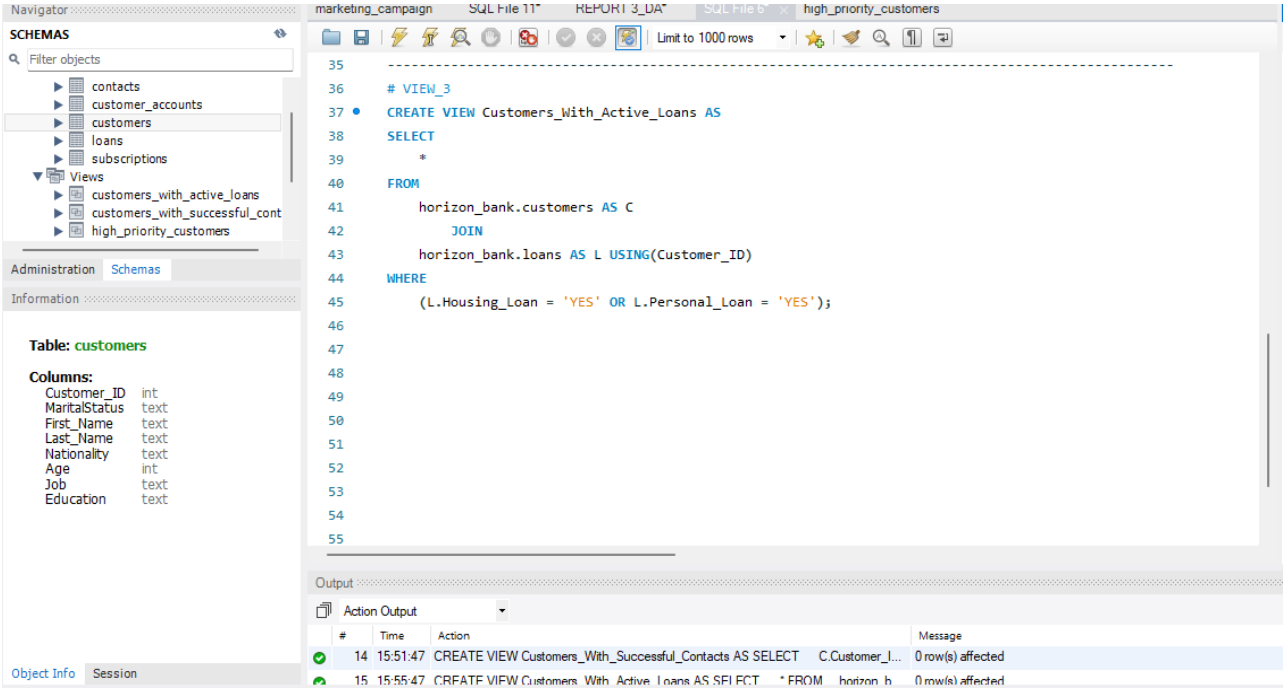
Access for End Users:

To facilitate easy access for the contact center team, North Star Consultants will implement user-friendly dashboards or reports that highlight customers with successful contacts. These can be integrated into the existing tools used by the contact center team for seamless access.

VIEW_3: Customers_With_Active_Loans

Description:

The Customers_With_Active_Loans view identifies customers with existing loans, aligning with the bank's goal of contacting them to explore opportunities for increased cash flow and eligibility for the new term deposit product. It was realized from phase 2 that only 4 people are categorized as top priority for the bank’s new term deposit product due to their huge account balance. Hence by identifying customers with loans (which is a requirement for accessing the new term deposit product), the bank can engage them to ensure they increase their account balance and qualify for the new product.



The view's benefit to Horizon Bank:

Loan Portfolio Optimization and Customer targeting: This view aids in efficiently managing the loan portfolio by identifying customers with active loans. It also allows the bank to strategically engage with these customers to increase their balances and qualify for the new term deposit product.

Access for End Users:

For the bank's engagement team, including loan officers and contact center representatives, North Star Consultants will implement accessible reporting tools. These tools may include automated reports or dashboards that highlight customers with active loans, ensuring that engagement teams can easily identify and reach out to these customers.

2. Report on Queries Linked to Views for Horizon Bank Optimization

Introduction:

As part of the ongoing initiative to optimize Horizon Bank's marketing strategies and customer

engagement, two queries have been developed based on the previously established views. These queries extract valuable insights from the identified views, providing actionable information for the bank's management.

Query_1: Engagement Strategy for Top Priority Customers

Purpose:

The first query, derived from the High_Priority_Customers view, aims to identify the top three priority customers for engagement by the bank manager. These customers have been categorized as "High_Priority" based on their account balances.

Query Script:

```sql
#QUERY_1
WITH top_3_priority_customers AS
(
SELECT
    *,
    Balance AS Account_Balance,
    ROW_NUMBER() OVER (PARTITION BY Priority_Status ORDER BY Balance DESC) AS RN
FROM
    horizon_bank.high_priority_customers AS P
    LEFT JOIN customer_accounts AS Acc USING (Customer_ID)
)
SELECT * FROM top_3_priority_customers
WHERE RN <= 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_ID	First_Name	Last_Name	Priority_Status	Balance	Account_Balance	RN
641686	Novah	Pierce	High_Priority	26306	26306	1
550331	Colten	Dean	High_Priority	23663	23663	2
671603	Cora	Velez	High_Priority	22856	22856	3

Interpretation:

The query identifies the top three priority customers, allowing the bank manager to contact three customers each day. The row numbers (RN) provide a sequence for engagement. For instance,

the customer with the highest balance is $26,306, making them the top-priority customer for immediate engagement.

Suggestions:

To enhance the engagement process, the bank manager could use this information to prioritize their outreach efforts. Additionally, considering the high balance of the top-priority customers, the bank may explore tailored financial products or services to further meet the needs of these valued customers.

Query_2: Analyzing Customer Contact Frequency by Age Group

Purpose:

The second query, derived from the Customers_With_Successful_Contacts view, analyzes the contact frequency of customers, grouped by age. The goal is to provide insights into the effectiveness of customer engagement across different age groups.

Query Script:

```sql
#QUERY_2
WITH Contact_Frequency AS (
    SELECT
        ROUND(Age / 10) * 10 AS Age_Group,
        COUNT(*) AS Contact_Count
    FROM Customers_With_Successful_Contacts AS CS
    LEFT JOIN customers AS C USING (Customer_ID)
    GROUP BY ROUND(Age / 10) * 10
)
SELECT
    Age_Group,
    SUM(Contact_Count) OVER (PARTITION BY Age_Group) AS Total_Contacts
FROM Contact_Frequency
ORDER BY Contact_Count DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Age_Group	Total_Contacts
40	36
30	35
50	27
60	17
80	6
70	5
20	1

Interpretation:

The query results provide a breakdown of successful customer contacts by age group. The age groups are rounded to the nearest decade, allowing for a comprehensive analysis of contact frequency.

Suggestions:

Targeted Outreach: Recognizing that the age group '40' has the highest contact count, the bank may consider targeted marketing campaigns or personalized offers for customers in this age bracket.

Engagement Strategies: For age groups with lower contact counts, the bank could explore innovative engagement strategies to increase outreach effectiveness.

Customer Segmentation: Further analysis of customer behaviors within each age group could lead to more precise customer segmentation and tailored communication strategies.

3. Indexes: Create and use indexes (on one column and on multiple columns)

Introduction:

Three index sets have been created to increase the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain them. The index and query will be tested on a table sample of 595 rows.

The purpose of the index is to increment the speed of results to identify customers with personal loan, housing_loan, and both to determine their likelihood to sign up for another loan product. In addition, this information will help the bank customer account team to priority which customers they should contact.

Once-Column Index:

```
53 •   CREATE INDEX index_personal ON `bank _campaigns`(personal_loan(10));
```

This query creates an index named index_personal on the personal_loan column of the bank_campaign table.

```
55 •   CREATE INDEX index_housing ON `bank _campaigns`(housing_loan(10));
```

This query creates an index named index_housing on the personal_housing column of the bank_campaign table.

The (10) after personal_loan and housing_loan suggests that this is a prefix index, which means the index is created on the first 10 characters of the personal_loan and housing_loan columns.

Two-Column Index:

```
CREATE INDEX index_personal_housing ON `bank _campaigns`(personal_loan(10),housing_loan(10));
```

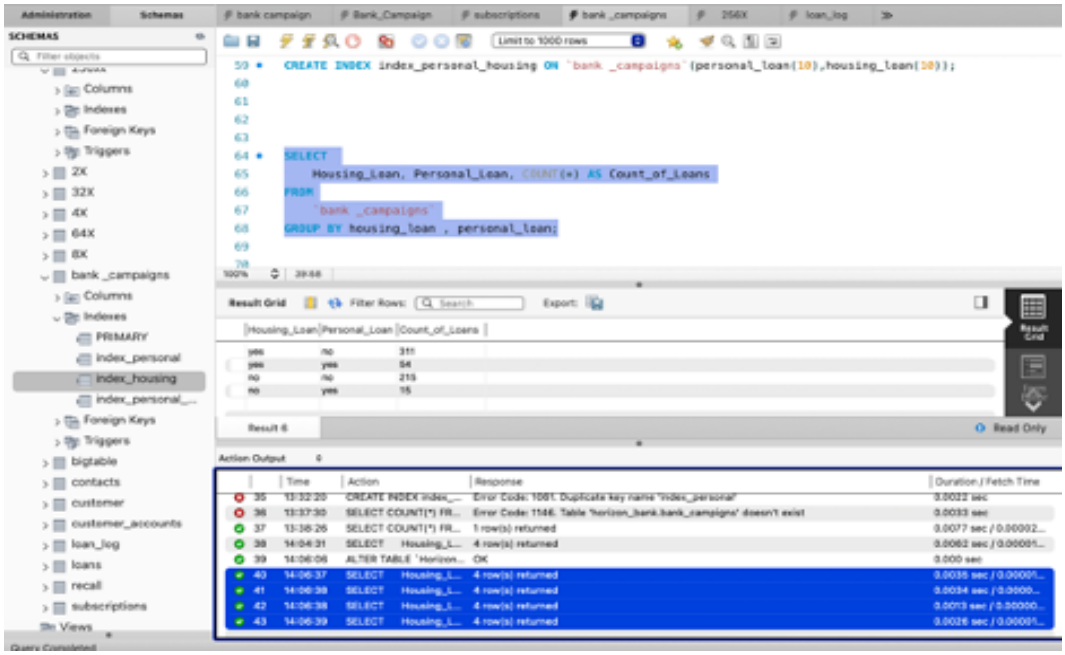
This query creates a composite index named index_personal_housing on two columns: personal_loan and housing_loan of the bank_campaign table. This composite index is useful when queries frequently filter or sort on these columns together.

3.a Speed Comparison for a complex query on a joint of small tables (with and without

index)

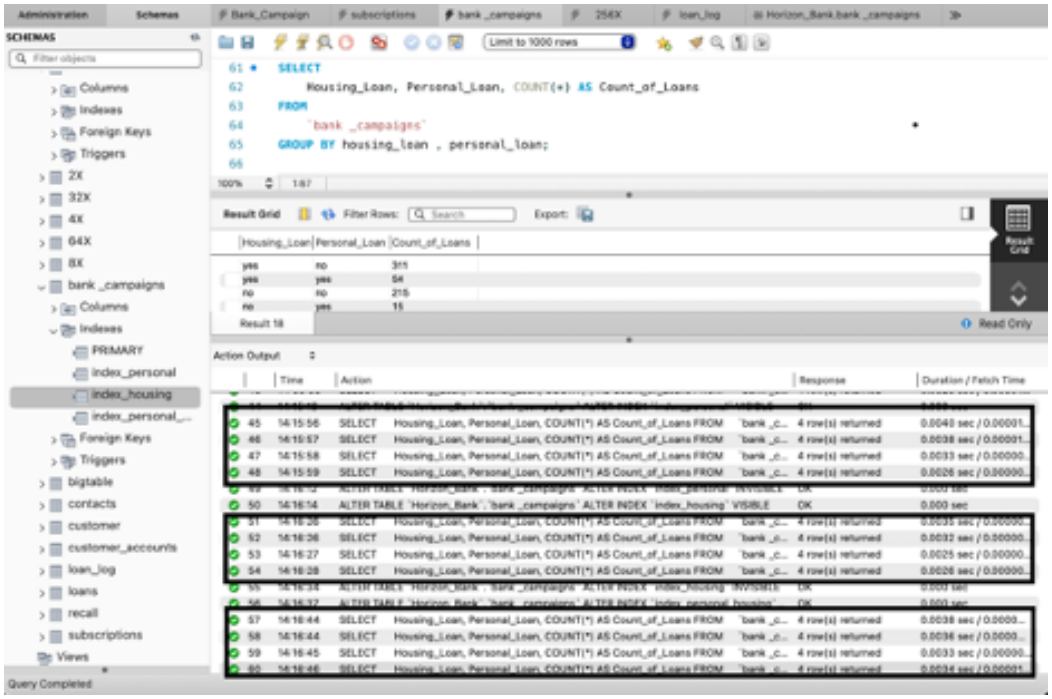
To measure and compare the efficiency of the index created, we will test a query in the small table with the index and without the index.

Query without index:



The speed of the results of the query without index has an average of 0.0027 sec.

Query with Index:



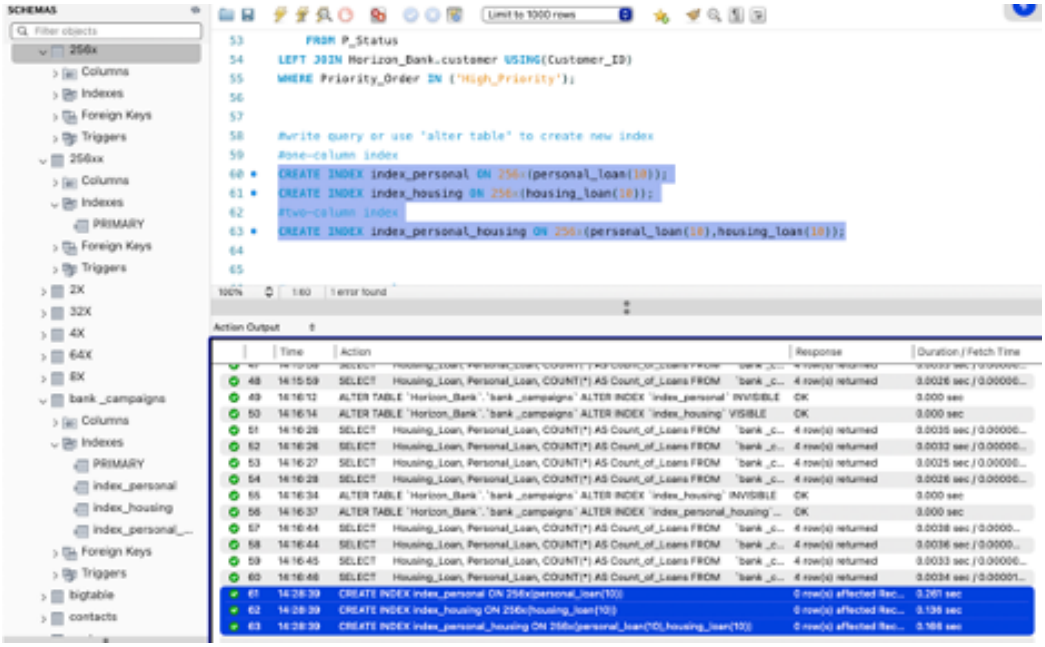
The queries with index have shown the following results:

- The speed of the results of the query with index_personal has an average of 0.0034 sec. (30% efficiency degradation in speed).
- The speed of the results of the query with index_housing has an average of 0.0029 sec. (12% efficiency degradation in speed).
- The speed of the results of the query with index_personal_housing has an average of 0.0035 sec. (34% efficiency degradation in speed).

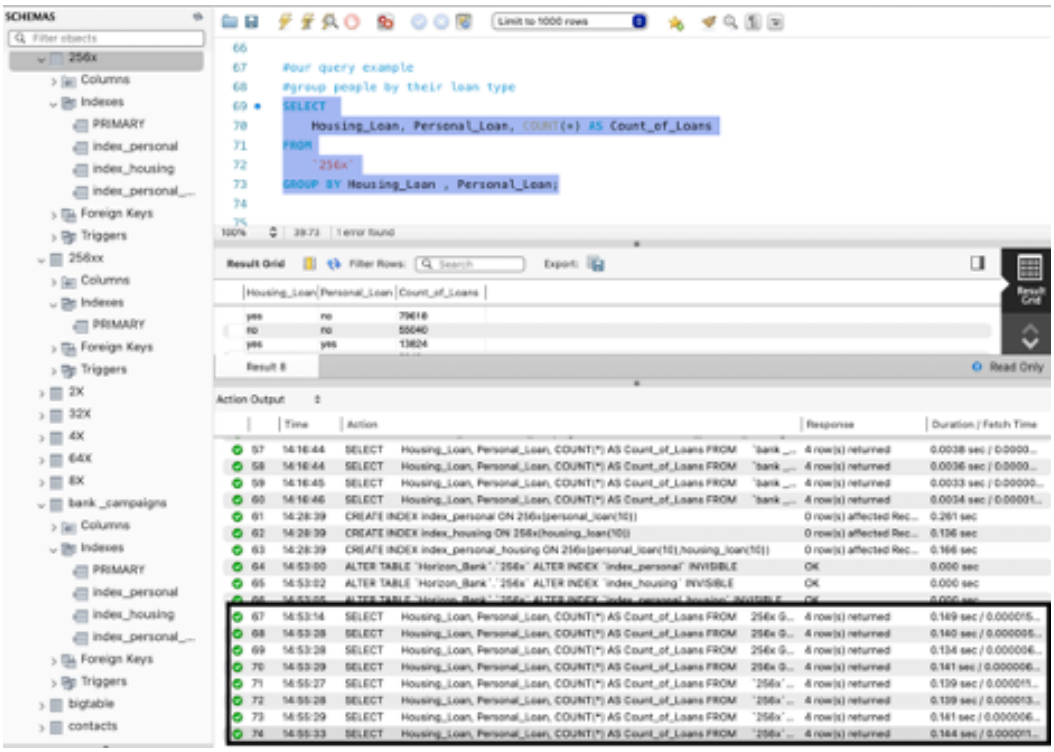
3.b Speed Comparison for a complex query on a joint of larger tables (with and without index)

To test if the results of the queries using the index are more efficient (fast) in a larger table and differ from results of a small table, we have included a larger table with 152,000 rows (table

‘256X’).

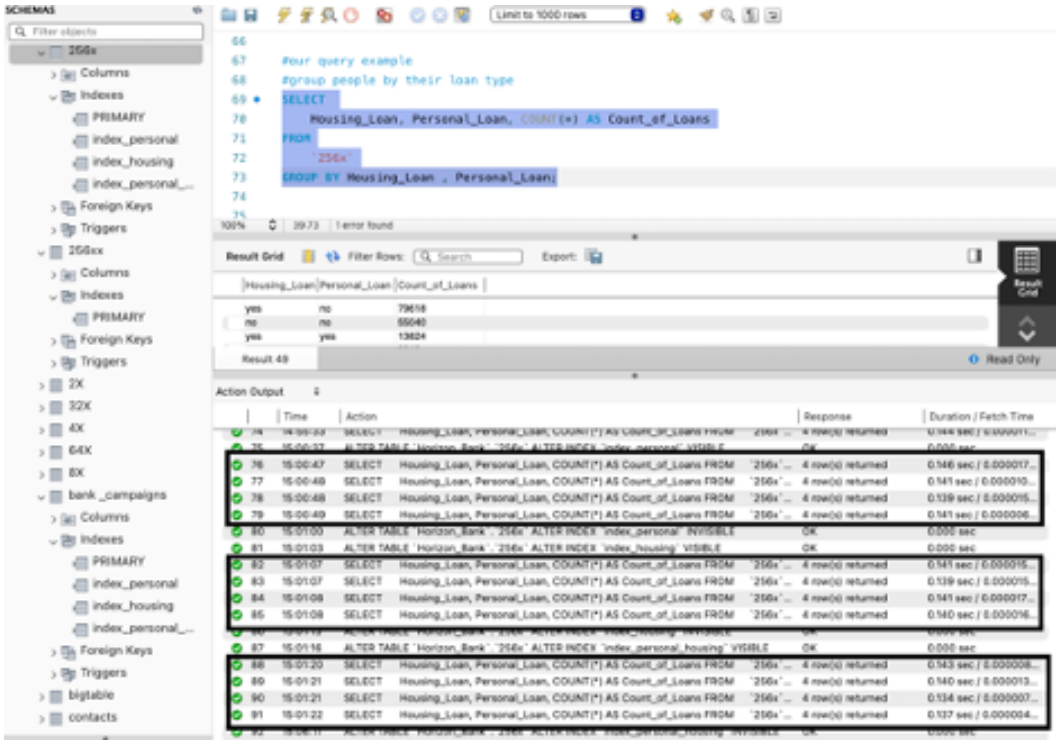


Query without Index:



The speed of the results of the query without index has an average of 0.140 sec.

Query with Index:



- The speed of the results of the query with index_personal has an average of 0.141 sec. (.07% efficiency degradation of speed).
- The speed of the results of the query with index_housing has an average of 0.140sec. (No changes in speed).
- The speed of the results of the query with index_personal_housing has an average of 0.138 sec. (1.42%efficiency improvement of speed).

Results:

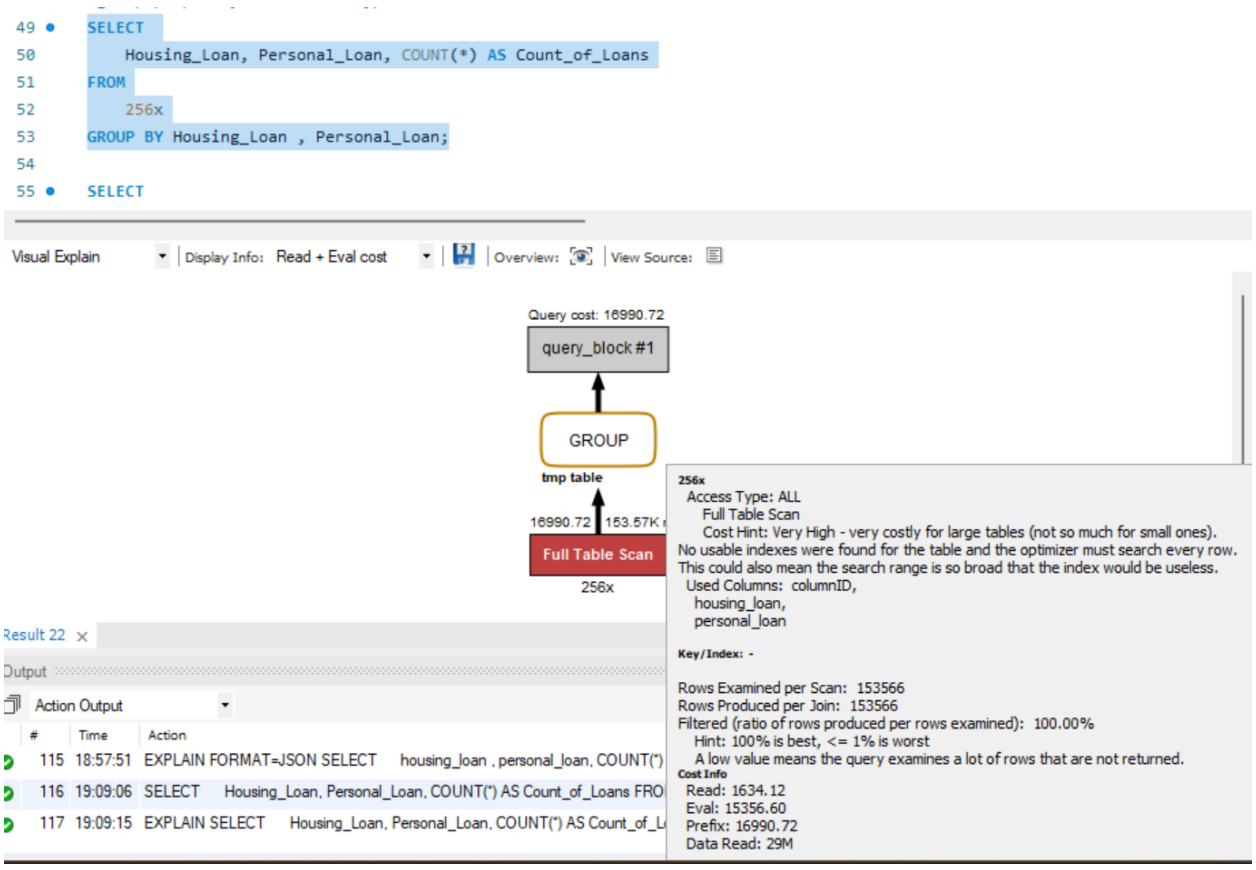
Effectiveness of Indexing: The indexes seem to have a more pronounced effect in larger tables. While the differences in execution times are relatively small in the small table sample, they

become more noticeable in the larger table.

One-Column vs. Two-Column Indexes: The two-column composite index (index_personal_housing) appears to provide a slight performance benefit in the larger table. This suggests that composite indexes can be particularly effective for queries that involve multiple fields.

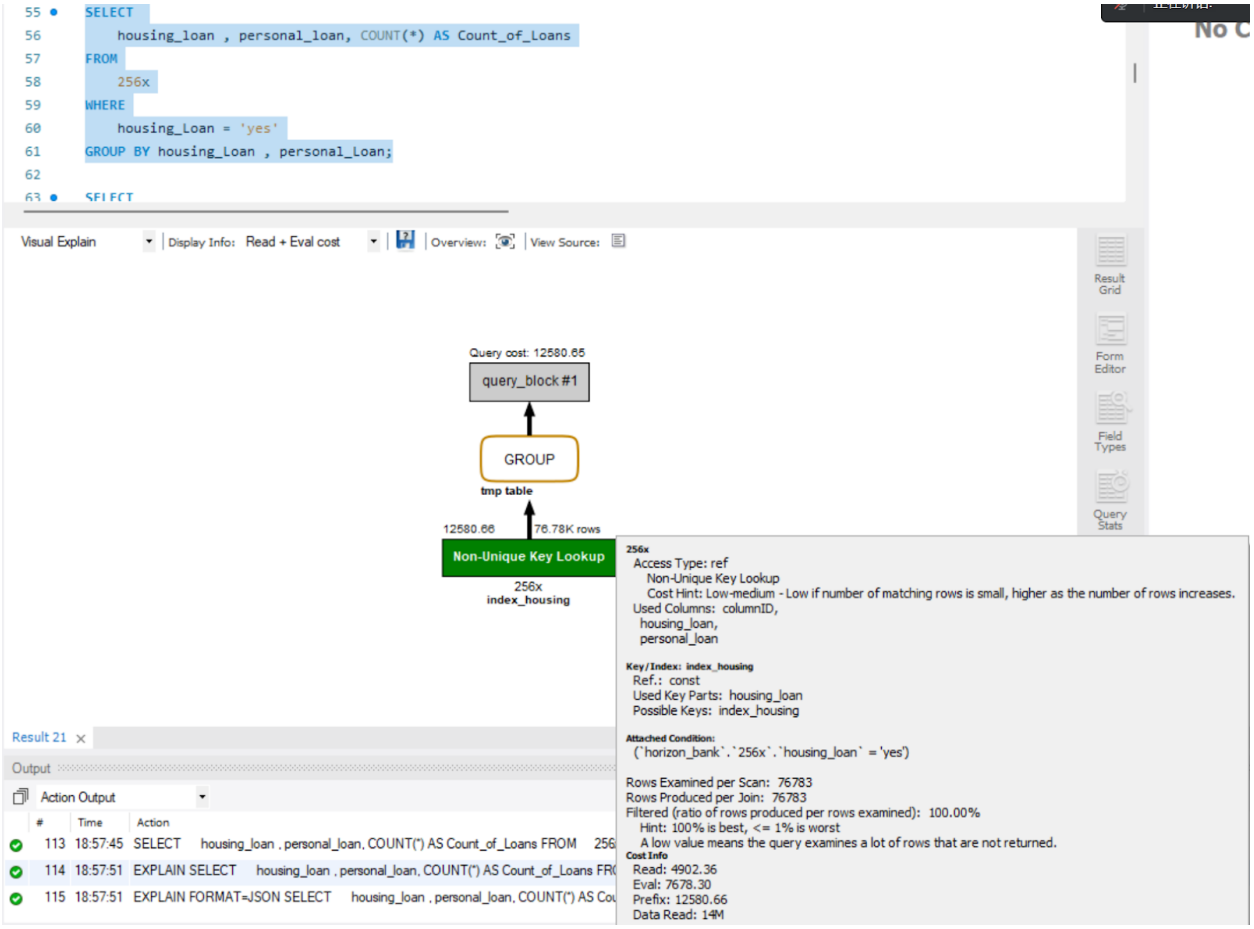
Prefix Indexing: Using prefix indexes might have impacted the performance differently depending on the distribution of data in the personal_loan and housing_loan columns. Prefix indexing is a trade-off between index size and query performance.

4. Explainer Diagram

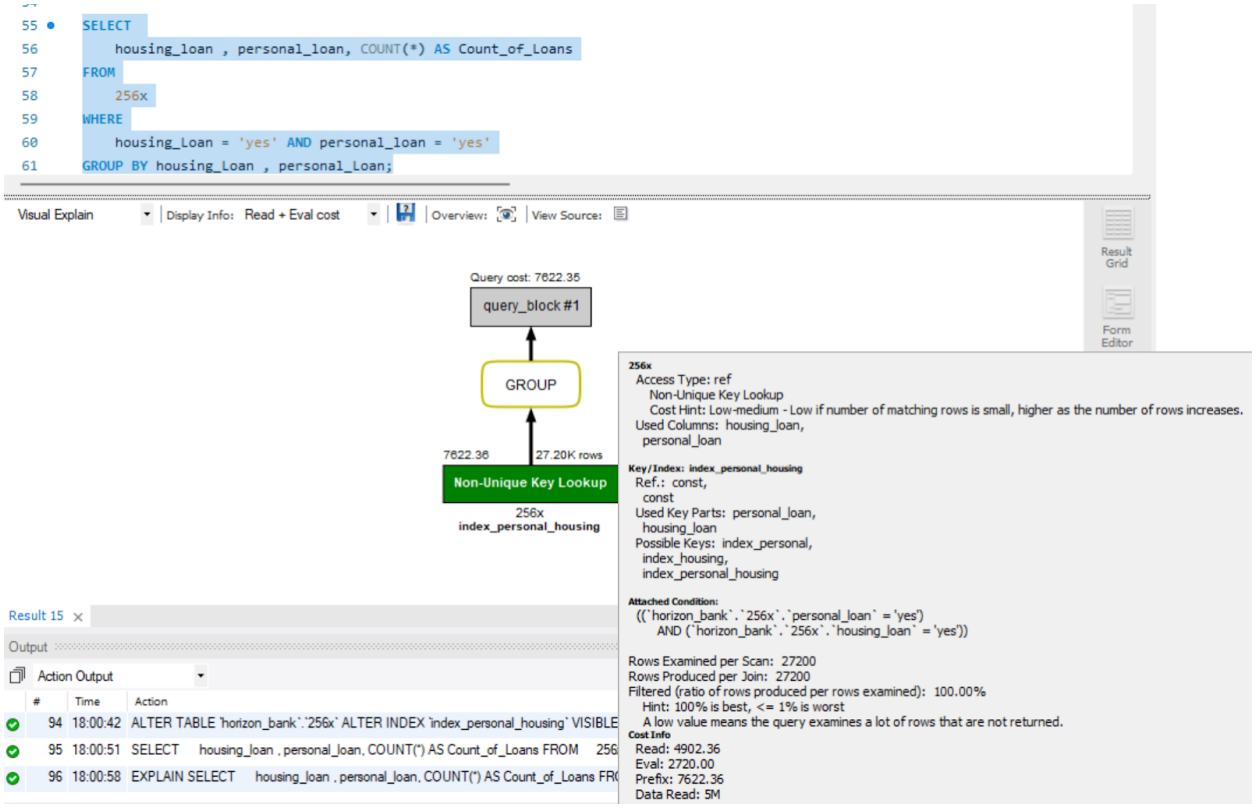


In the first screenshot, The query has a high cost, indicating its inefficiency. It is performing a full table scan, meaning it is examining every row(153.58k rows) in the table rather than using

indexes to narrow down the search, which is typically slower.



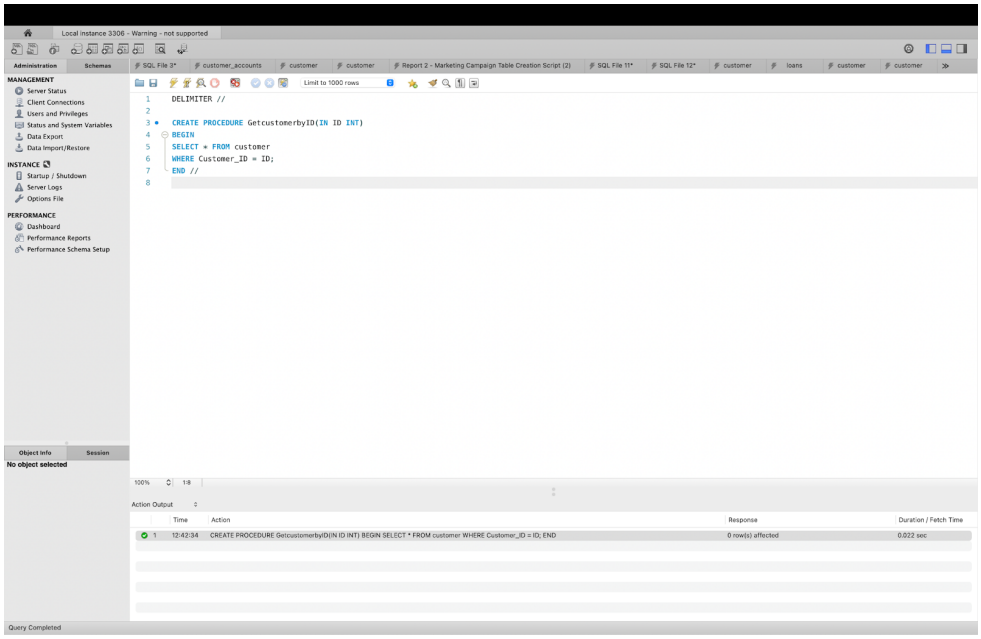
In the second screenshot, the clause is modified by adding a where function to activate the index. The index being used is index_housing, which is based on the housing_loan column in the original dataset. In this result, The number of rows examined per scan has been decreased to 86.7k rows, the efficiency has been raised successfully, but needs further improvement.



In the third screenshot, by modifying the where clause again, this query can use the two-column index created before successfully. The query utilizes a composite index :index_personal_housing. This two-column index allows the query to reference these two fields quickly, significantly reducing the number of rows that need to be examined. However, since it still involves checking 27.2k rows(the original one is 153.57k rows, and the one-column index is 86.7k rows), it suggests that the index may still not be optimal, or the query conditions may not entirely align with the best use scenario for the index.

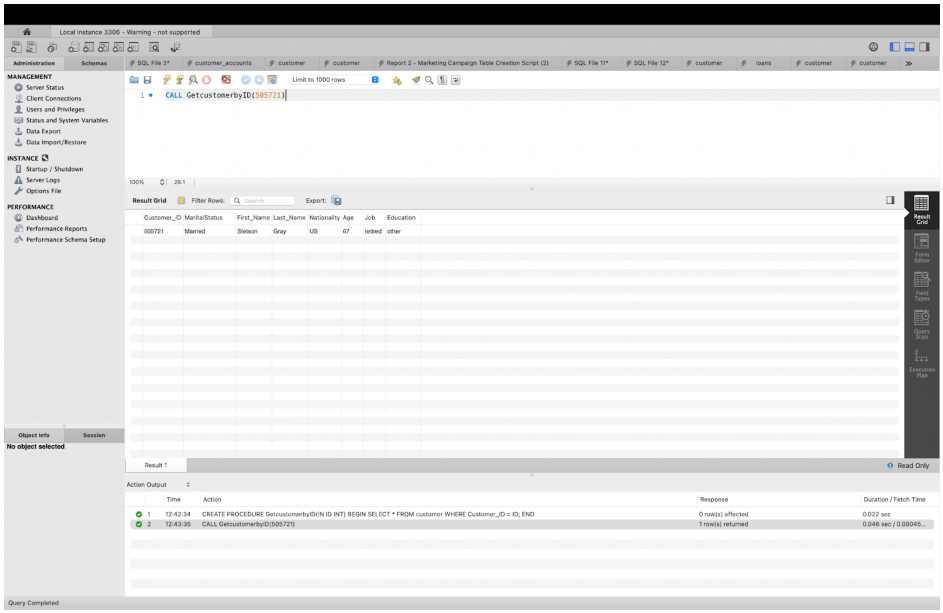
5. Stored Procedures: three (3) different Stored Procedures using different parameters.

Stored Procedure 1: Get customer by ID

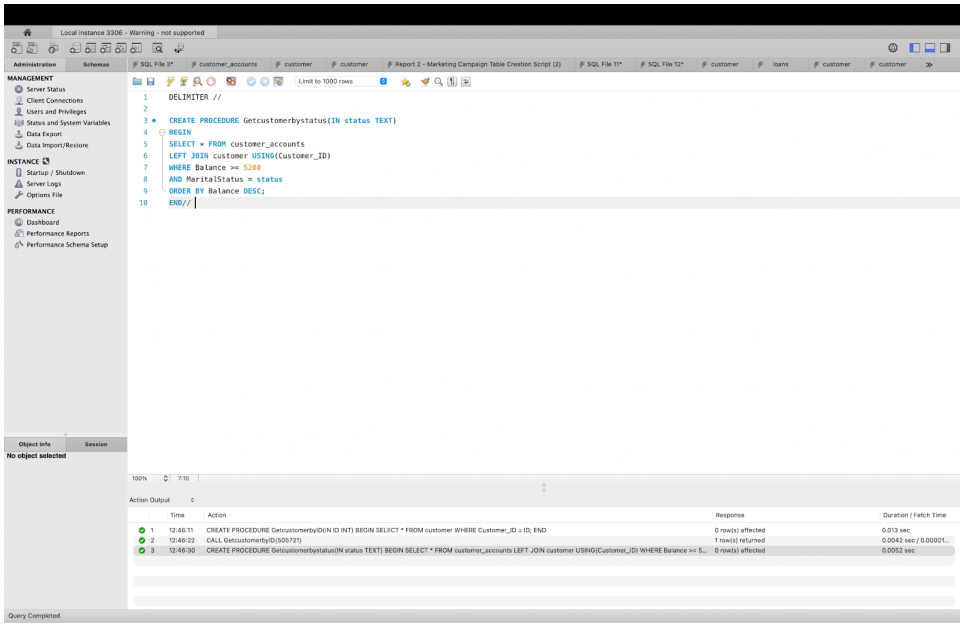


The SQL code above defines a stored procedure named 'GetcustomerbyID' that retrieves customer information from the "customer" table based on the specific customer 'ID'. The procedure takes an input parameter ID, and when called with a customer value, in this case 505721, using 'CALL GetcustomerID(505721)', it executes the SELECT query, fetching all columns for the customer matching Customer_ID (505721).

Result:



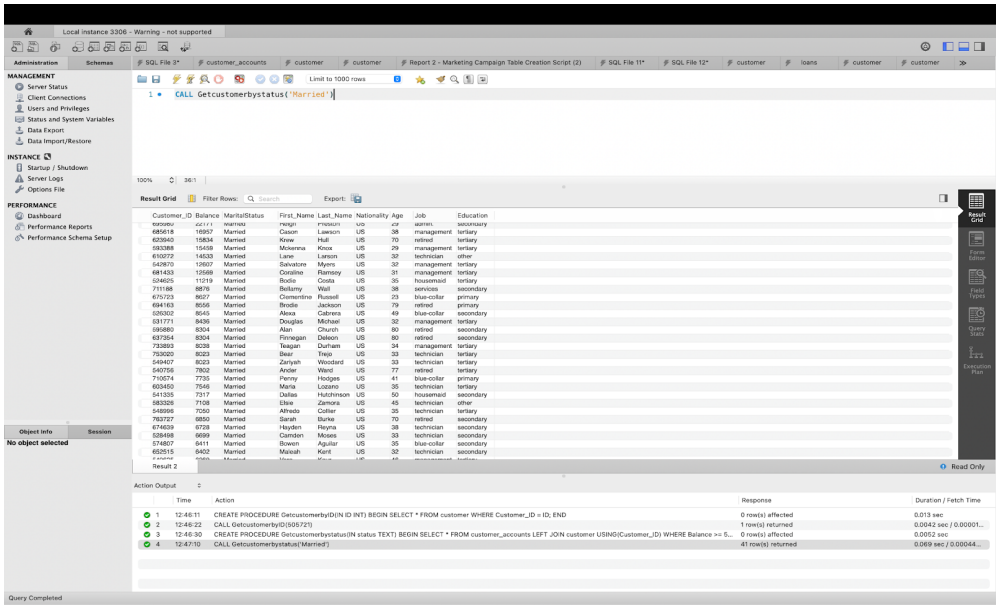
Stored Procedure 2: Get customer by Status



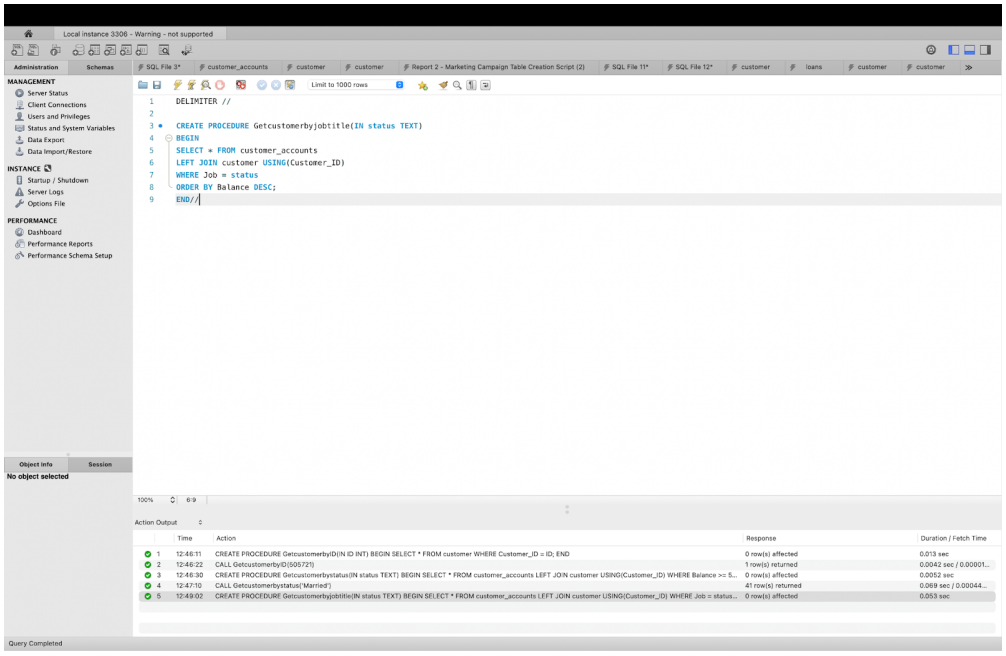
The SQL code above defines a stored procedure named ‘Getcustomerbystatus’ that retrieves customer information from the “customer accounts” table, joined with the “customer” table, based on a specified marital status and a minimum balance requirement. The procedure takes an input parameter “status”, and when called, it executes the SELECT query, fetching all columns for customers with a balance greater than or equal to 5200 and matching the provided marital

status. The result is ordered by balance in descending order. The stored procedure is then called with the argument “Married” to fetch relevant customer information.

Result:

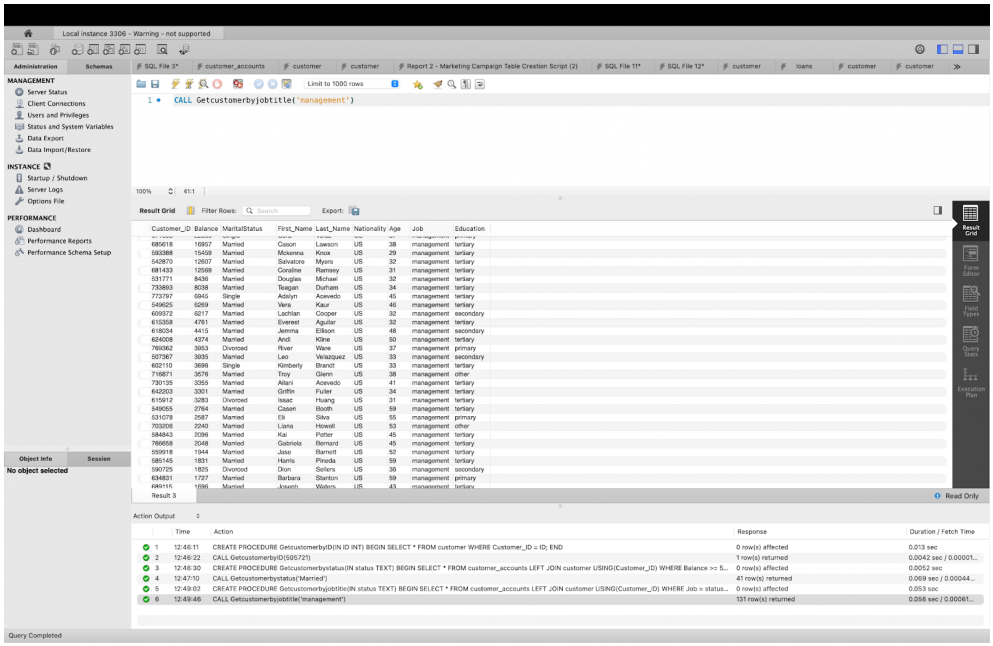


Stored Procedure 3: Get customer by Job Title



The SQL code above creates a stored procedure names “Getcustomerbyjobtitle” that retrieves customer information from the “customer_accounts” table, joined with the “customer” table. It filters the result based on a specific job title (“management”) and orders it by balance in descending order. The stored procedure is then called the argument ‘management’ to fetch relevant customer information.

Result:



6. Triggers: three (3) different types of triggers: insert, update, and delete.

Trigger1: After Insert

```
#trigger: pick those people who may need another telephone campaign
#(those who first telephone duration is under 10s)
CREATE TABLE recall (
    recall_id INT AUTO_INCREMENT PRIMARY KEY,
    action_type VARCHAR(50),
    duration_second INT,
    subscribe_result VARCHAR(255),
    action_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    whether_recall VARCHAR(255)
);

DELIMITER //

CREATE TRIGGER 256x_AFTER_INSERT
AFTER INSERT ON `256x`
FOR EACH ROW
BEGIN
    IF NEW.duration_second < 10 THEN
        INSERT INTO recall (recall_id, duration_second, subscribe_result, whether_recall)
        VALUES (NEW.columnID, NEW.duration_second, NEW.subscribe_result, 'yes');
    END IF;
END;
//

DELIMITER ;
```

```
#insert
INSERT INTO 256x (columnID, duration_second, subscribe_result)
VALUES ('250080','7', 'no');

INSERT INTO 256x (columnID, duration_second, subscribe_result)
VALUES ('289023','5', 'no');

INSERT INTO 256x (columnID, duration_second, subscribe_result)
VALUES ('289899','9', 'no');
```

Query Description:

This query is designed to identify customers whose initial telephone campaign duration is less than ten seconds and categorize them as 'people in need of a second call.' They will be added to a new table named 'recall.'

First, I created the 'recall' table. Then, I establish an after-insert trigger. This trigger activates

when new rows are inserted into the '256x' table (which contains comprehensive customer information). If the call duration in the newly inserted rows is under 10 seconds, three pieces of personal information, including 'columnID,' 'duration,' and 'subscribe result,' will also be inserted into the 'recall' table. Finally, I insert three sample records to test whether this trigger runs correctly.

Result:

```
1 •   SELECT * FROM horizon_bank.recall;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

recall_id	duration_second	subscribe_result	action_timestamp	whether_recall
250080	7	no	2023-11-08 04:52:03	yes
289023	5	no	2023-11-08 04:53:15	yes
289899	9	no	2023-11-08 16:07:27	yes
NULL	NULL	NULL	NULL	NULL

Trigger 2: Before Update

```sql
# trigger 2: find out those people who may need a third call and record them.
#(second_durataion > 10s and subscribe_result_2 = 'no')

#add a column 'recall_situation' to recall table
ALTER TABLE recall
ADD second_duration INT,
ADD subscribe_result_2 VARCHAR(10),
ADD whether_third_recall VARCHAR(10);

DELIMITER //
CREATE TRIGGER BEFORE_RECALL_UPDATE BEFORE UPDATE ON recall
FOR EACH ROW
BEGIN
  IF NEW.second_duration < 10 THEN
    SET NEW.whether_third_recall = 'no';
  ELSEIF NEW.second_duration >= 10 AND NEW.subscribe_result_2 = 'no' THEN
    SET NEW.whether_third_recall = 'yes';
  ELSEIF NEW.subscribe_result_2 = 'yes' THEN
    SET NEW.whether_third_recall = 'no';
  END IF;
END;
//
DELIMITER ;
```

```
125 •   # update recall table
126     UPDATE recall
127     SET
128     second_duration = 15 ,
129     subscribe_result_2 = 'no'
130     WHERE
131     recall_id = 250080;
132
133 •   UPDATE recall
134     SET
135     second_duration = 3,
136     subscribe_result_2 = 'no'
137     WHERE
138     recall_id = 289023;
139
140 •   UPDATE recall
141     SET
142     second_duration = 121,
143     subscribe_result_2 = 'yes'
144     WHERE
145     recall_id = 289899;
```

Query Description:

This query is designed to identify customers whose second telephone campaign duration is over 10 seconds but still not subscribe for the product. They will be automatically marked as ‘people need a third call’, and the others should be marked as ‘people do not need a third call’ in the ‘recall’ table.

First, I created three new columns for the recall table to record the result of the second telephone campaign. Then, I establish a before-update trigger. This trigger is activated when some information is inserted into those three new columns. When the second telephone campaign duration(second_duration) is over 10 seconds and the second subscription result(subscribe_result_2) is still ‘no’, the ‘whether_third_recall’ will be input by ‘yes’ automatically. Finally, I update three sample records to test whether this trigger runs correctly.

Result:

```
1 •  SELECT * FROM horizon_bank.recall;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

recall_id	duration_second	subscribe_result	action_timestamp	whether_recall	second_duration	subscribe_result_2	whether_third_recall
250080	7	no	2023-11-08 04:52:03	yes	15	no	yes
289023	5	no	2023-11-08 04:53:15	yes	3	no	no
289899	9	no	2023-11-08 16:07:27	yes	121	yes	no
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Trigger 3: Before Delete

```
148     #trigger 3: delete and record those unpotential customers of this product
149     # those who subscribe_result_2 and whether_third_recall are both 'no'
150
151 •   CREATE TABLE unpotential_customer(
152     recall_id INT AUTO_INCREMENT PRIMARY KEY
153     );
154
155     DELIMITER //
156 •   CREATE TRIGGER before_recall_delete
157     BEFORE DELETE ON recall
158     FOR EACH ROW
159     BEGIN
160         IF OLD.subscribe_result_2 = 'no' AND OLD.whether_third_recall = 'no' THEN
161             INSERT INTO unpotential_customer SET recall_id = OLD.recall_id;
162         END IF;
163     END;
164     //
165     DELIMITER ;
166
167 •   #delete
168     DELETE FROM recall
169     WHERE subscribe_result_2 = 'no' AND whether_third_recall = 'no'
170     LIMIT 10;
```

Query Description:

This query is designed to identify those people who do not subscribe for the

product and also do not need a third call. Those customers will be marked as ‘potential customers’.

Firstly, I created the 'unpotential_customer' table. Then, a before delete trigger is defined, which is programmed to act before a deletion operation is executed on the recall table.It examines the row which meets specific criteria: both the subscribe_result_2 and whether_third_recall fields are set to 'no'. If the row meets these conditions, it triggers an action that inserts the recall_id from this row into an unpotential_customer table. Finally, I run a sample record to test whether this trigger runs correctly.

Result:

```
1 • SELECT * FROM horizon_bank.recall;
2 • SELECT * FROM unpotential_customer;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

recall_id	duration_second	subscribe_result	action_timestamp	whether_recall	second_duration	subscribe_result_2	whether_third_recall
250080	7	no	2023-11-08 04:52:03	yes	15	no	yes
289899	9	no	2023-11-08 16:07:27	yes	121	yes	no
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL