Retail Banking Data- CRM

Madhav Appaneni
University at Buffalo
Buffalo, New York, USA
madhavap@buffalo.edu

I. Introduction

The dataset consists of retail banking demo data. Banking is one such sector where databases are most useful since the transactions happen at a faster pace and all the updates need to happen in real time. Since the transactions happen in real time, the data that gets generated will be huge which makes it difficult to store and process in excel sheets. On the other hand, when a customer performs transactions on his/her account, it would not be ideal for both customer and financial institution if they get erroneous results. Further, databases support concurrency. In this case, customers, bank officials and customer support agents can seamlessly ingest and access data. Databases also help in keeping the data secure in an organization with multiple access and role support. For example, the customer support agents must not be allowed to edit information about a branch or a bank. Using a spreadsheet would not be an ideal solution for this use. In this case, we have data coming from many input sources, if we use a spreadsheet, we will have to duplicate parts of records for better readability. But with databases, we get to join multiple relations and access data with better flexibility To tackle all these issues, we aim to design a database which handles all CRUD operations in real time.

II. Target Audience

This database is used by three sets of people. 1. Bank officials: The bank officials use this database to store and retrieve information about the customers – their accounts, balances, addresses and other information 2. Customers: The customers use this database indirectly via a web interface to check their balance, transactions, addresses, card details and their customer support incidents. 3. Customer Support Agents: The customer support agents will use this database via a web interface or via CRM software to address customer queries about their accounts and raise any incidents or tickets if any customer faces any issues.

III. Database Administrator AND Users

The main users of the CRM Retail Banking dashboard will be the Customer support agents. When customers call the customer care center, they take the complaint from the customers. This dashboard will help to maintain and keep track of the complaints using complaint IDs of different customers including past records. The database administrators will be the IT personnel of the bank. They would be responsible for maintaining the integrity, optimizing the run times, and ensuring the sanity of the database.

IV. Database Structure

A. Intial ER Diagram

Entity relationship diagram is represented in Fig. 1 It shows the relation between different tables in the database.

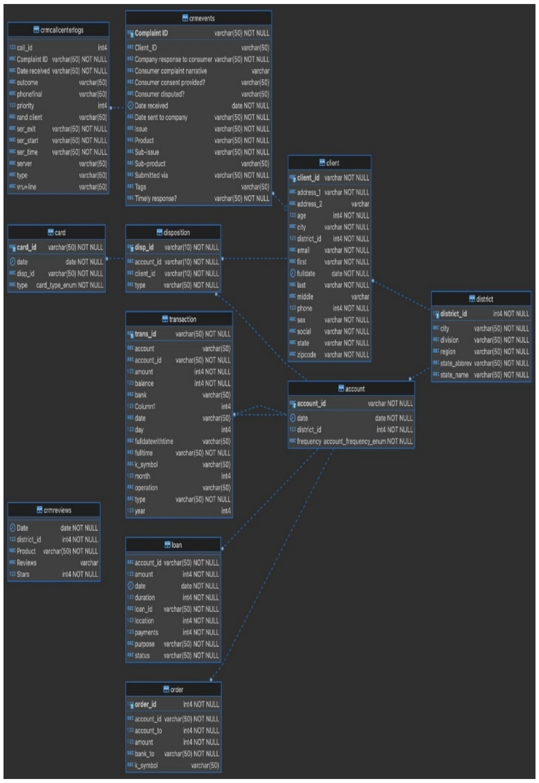
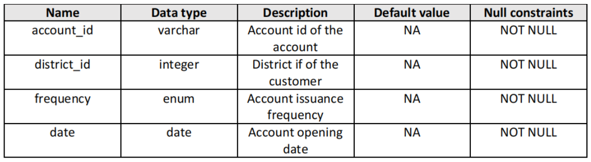

crmcallcenterlogs
call_id int4
Complaint ID varchar(50) NOT NULL
Date received varchar(50) NOT NULL
outcome varchar(50)
phonefinal varchar(50)
priority int4
rand client varchar(50)
ser_exit varchar(50) NOT NULL
ser_start varchar(50) NOT NULL
ser_time varchar(50) NOT NULL
server varchar(50)
type varchar(50)
vru+line varchar(50)
crmevents
Complaint ID varchar(50) NOT NULL
Client_ID varchar(50)
Company response to consumer varchar(50) NOT NULL
Consumer complaint narrative varchar
Consumer consent provided? varchar(50)
Consumer disputed? varchar(50)
Date received date NOT NULL
Date sent to company varchar(50) NOT NULL
Issue varchar(50) NOT NULL
Product varchar(50) NOT NULL
Sub-issue varchar(50) NOT NULL
Sub-product varchar(50)
Submitted via varchar(50) NOT NULL
Tags varchar(50)
Timely response? varchar(50) NOT NULL
client
client_id varchar NOT NULL
address_1 varchar NOT NULL
address_2 varchar
age int4 NOT NULL
city varchar NOT NULL
district_id int4 NOT NULL
email varchar NOT NULL
first varchar NOT NULL
fulldate date NOT NULL
last varchar NOT NULL
middle varchar
phone int4 NOT NULL
sex varchar NOT NULL
social varchar NOT NULL
state varchar NOT NULL
zipcode varchar NOT NULL
card
card_id varchar(50) NOT NULL
date date NOT NULL
disp_id varchar(50) NOT NULL
type card_type_enum NOT NULL
disposition
disp_id varchar(10) NOT NULL
account_id varchar(10) NOT NULL
client_id varchar(10) NOT NULL
type varchar(50) NOT NULL
district
district_id int4 NOT NULL
city varchar(50) NOT NULL
division varchar(50) NOT NULL
region varchar(50) NOT NULL
state_abbrev varchar(50) NOT NULL
state_name varchar(50) NOT NULL
transaction
trans_id varchar(50) NOT NULL
account varchar(50)
account_id varchar(50) NOT NULL
amount int4 NOT NULL
balance int4 NOT NULL
bank varchar(50)
Column1 int4
date varchar(50)
day int4
fulldatewithtime varchar(50)
fulltime varchar(50) NOT NULL
k_symbol varchar(50)
month int4
operation varchar(50)
type varchar(50) NOT NULL
year int4
account
account_id varchar NOT NULL
date date NOT NULL
district_id int4 NOT NULL
frequency account_frequency_enum NOT NULL
crmreviews
Date date NOT NULL
district_id int4 NOT NULL
Product varchar(50) NOT NULL
Reviews varchar
Stars int4 NOT NULL
loan
account_id varchar(50) NOT NULL
amount int4 NOT NULL
date date NOT NULL
duration int4 NOT NULL
loan_id varchar(50) NOT NULL
location int4 NOT NULL
payments int4 NOT NULL
purpose varchar(50) NOT NULL
status varchar(50) NOT NULL
order
order_id int4 NOT NULL
account_id varchar(50) NOT NULL
account_to int4 NOT NULL
amount int4 NOT NULL
bank_to varchar(50) NOT NULL
k_symbol varchar(50)

Fig 1: ER Diagram

B. Relations

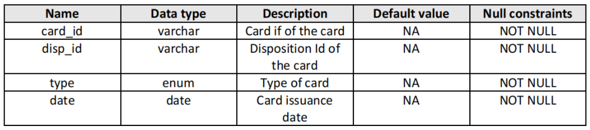
- we have 11 relations in the database.The details of the relations are described as below
- **Account:**



Name	Data type	Description	Default value	Null constraints
account_id	varchar	Account id of the account	NA	NOT NULL
district_id	integer	District if of the customer	NA	NOT NULL
frequency	enum	Account issuance frequency	NA	NOT NULL
date	date	Account opening date	NA	NOT NULL

Fig 2: Schema of Account table

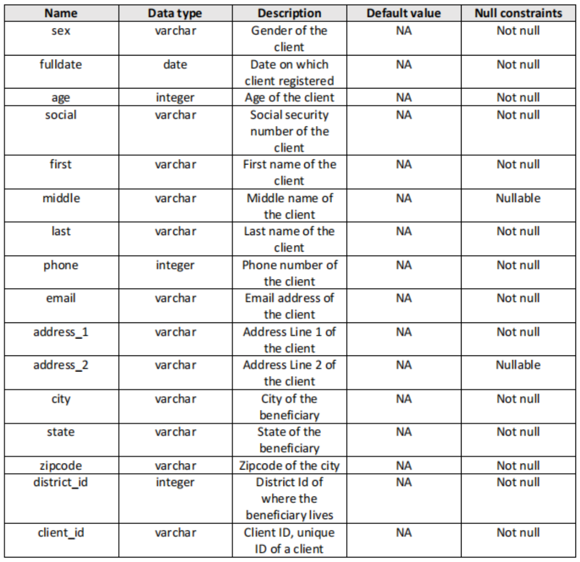
- **Card:**



Name	Data type	Description	Default value	Null constraints
card_id	varchar	Card if of the card	NA	NOT NULL
disp_id	varchar	Disposition Id of the card	NA	NOT NULL
type	enum	Type of card	NA	NOT NULL
date	date	Card issuance date	NA	NOT NULL

Fig 3: Schema of Card Table

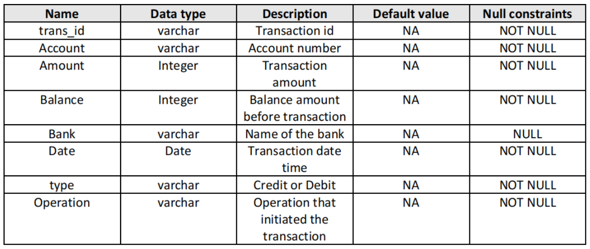
- **Client:**



Name	Data type	Description	Default value	Null constraints
sex	varchar	Gender of the client	NA	Not null
fulldate	date	Date on which client registered	NA	Not null
age	integer	Age of the client	NA	Not null
social	varchar	Social security number of the client	NA	Not null
first	varchar	First name of the client	NA	Not null
middle	varchar	Middle name of the client	NA	Nullable
last	varchar	Last name of the client	NA	Not null
phone	integer	Phone number of the client	NA	Not null
email	varchar	Email address of the client	NA	Not null
address_1	varchar	Address Line 1 of the client	NA	Not null
address_2	varchar	Address Line 2 of the client	NA	Nullable
city	varchar	City of the beneficiary	NA	Not null
state	varchar	State of the beneficiary	NA	Not null
zipcode	varchar	Zipcode of the city	NA	Not null
district_id	integer	District Id of where the beneficiary lives	NA	Not null
client_id	varchar	Client ID, unique ID of a client	NA	Not null

Fig 4: Schema of client Table

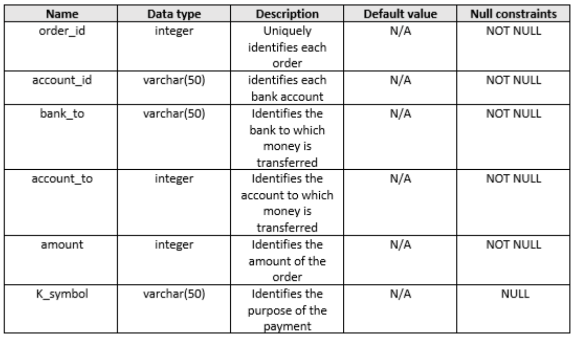
- **Transaction:**



Name	Data type	Description	Default value	Null constraints
trans_id	varchar	Transaction id	NA	NOT NULL
Account	varchar	Account number	NA	NOT NULL
Amount	Integer	Transaction amount	NA	NOT NULL
Balance	Integer	Balance amount before transaction	NA	NOT NULL
Bank	varchar	Name of the bank	NA	NULL
Date	Date	Transaction date time	NA	NOT NULL
type	varchar	Credit or Debit	NA	NOT NULL
Operation	varchar	Operation that initiated the transaction	NA	NOT NULL

Fig 5: Schema of transaction table

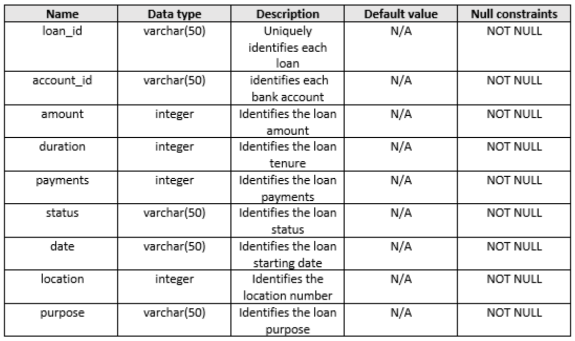
- **order:**



Name	Data type	Description	Default value	Null constraints
order_id	integer	Uniquely identifies each order	N/A	NOT NULL
account_id	varchar(50)	identifies each bank account	N/A	NOT NULL
bank_to	varchar(50)	Identifies the bank to which money is transferred	N/A	NOT NULL
account_to	integer	Identifies the account to which money is transferred	N/A	NOT NULL
amount	integer	Identifies the amount of the order	N/A	NOT NULL
K_symbol	varchar(50)	Identifies the purpose of the payment	N/A	NULL

Fig 6: Schema of order Table

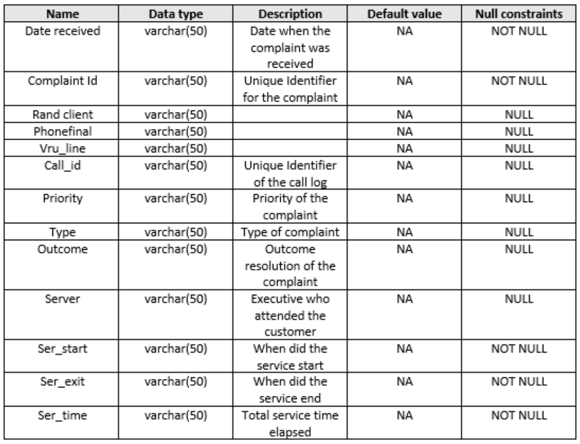
- **Loan:**



Name	Data type	Description	Default value	Null constraints
loan_id	varchar(50)	Uniquely identifies each loan	N/A	NOT NULL
account_id	varchar(50)	identifies each bank account	N/A	NOT NULL
amount	integer	Identifies the loan amount	N/A	NOT NULL
duration	integer	Identifies the loan tenure	N/A	NOT NULL
payments	integer	Identifies the loan payments	N/A	NOT NULL
status	varchar(50)	Identifies the loan status	N/A	NOT NULL
date	varchar(50)	Identifies the loan starting date	N/A	NOT NULL
location	integer	Identifies the location number	N/A	NOT NULL
purpose	varchar(50)	Identifies the loan purpose	N/A	NOT NULL

Fig 7: Schema of loan table

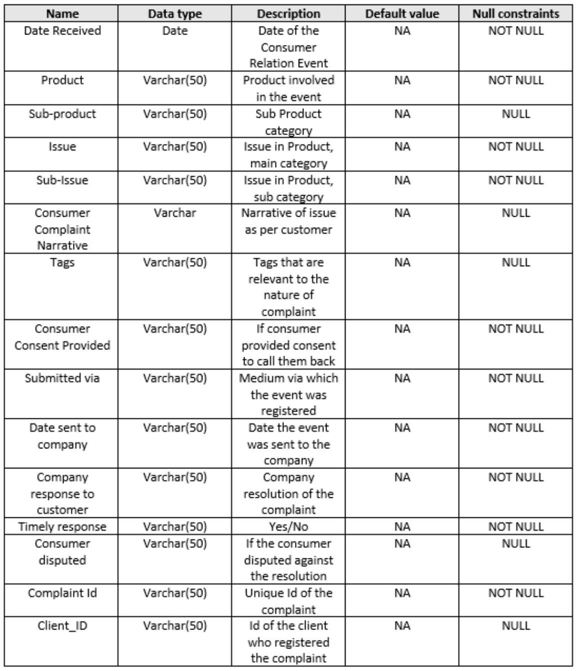
- **Call Center Logs:**



Name	Data type	Description	Default value	Null constraints
Date received	varchar(50)	Date when the complaint was received	NA	NOT NULL
Complaint Id	varchar(50)	Unique Identifier for the complaint	NA	NOT NULL
Rand client	varchar(50)		NA	NULL
Phonefinal	varchar(50)		NA	NULL
Vru_line	varchar(50)		NA	NULL
Call_id	varchar(50)	Unique Identifier of the call log	NA	NULL
Priority	varchar(50)	Priority of the complaint	NA	NULL
Type	varchar(50)	Type of complaint	NA	NULL
Outcome	varchar(50)	Outcome resolution of the complaint	NA	NULL
Server	varchar(50)	Executive who attended the customer	NA	NULL
Ser_start	varchar(50)	When did the service start	NA	NOT NULL
Ser_exit	varchar(50)	When did the service end	NA	NOT NULL
Ser_time	varchar(50)	Total service time elapsed	NA	NOT NULL

Fig 8: Schema of Call center logs Table

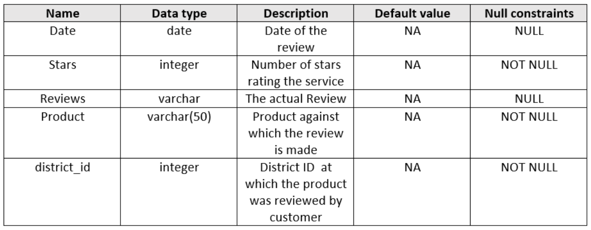
- **CRM Events:**



Name	Data type	Description	Default value	Null constraints
Date Received	Date	Date of the Consumer Relation Event	NA	NOT NULL
Product	Varchar(50)	Product involved in the event	NA	NOT NULL
Sub-product	Varchar(50)	Sub Product category	NA	NULL
Issue	Varchar(50)	Issue in Product, main category	NA	NOT NULL
Sub-Issue	Varchar(50)	Issue in Product, sub category	NA	NOT NULL
Consumer Complaint Narrative	Varchar	Narrative of issue as per customer	NA	NULL
Tags	Varchar(50)	Tags that are relevant to the nature of complaint	NA	NULL
Consumer Consent Provided	Varchar(50)	If consumer provided consent to call them back	NA	NOT NULL
Submitted via	Varchar(50)	Medium via which the event was registered	NA	NOT NULL
Date sent to company	Varchar(50)	Date the event was sent to the company	NA	NOT NULL
Company response to customer	Varchar(50)	Company resolution of the complaint	NA	NOT NULL
Timely response	Varchar(50)	Yes/No	NA	NOT NULL
Consumer disputed	Varchar(50)	If the consumer disputed against the resolution	NA	NULL
Complaint Id	Varchar(50)	Unique Id of the complaint	NA	NOT NULL
Client_ID	Varchar(50)	Id of the client who registered the complaint	NA	NULL

Fig 9: Schema of CRM Events Table

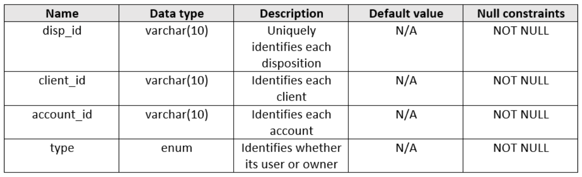
- **CRM Reviews:**



Name	Data type	Description	Default value	Null constraints
Date	date	Date of the review	NA	NULL
Stars	integer	Number of stars rating the service	NA	NOT NULL
Reviews	varchar	The actual Review	NA	NULL
Product	varchar(50)	Product against which the review is made	NA	NOT NULL
district_id	integer	District ID at which the product was reviewed by customer	NA	NOT NULL

Fig 11: Schema of CRM Reviews Table

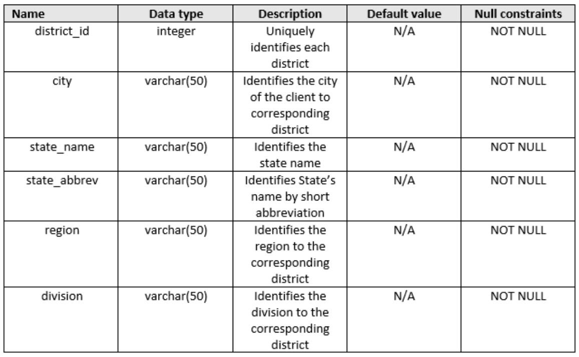
- **Disposition:**



Name	Data type	Description	Default value	Null constraints
disp_id	varchar(10)	Uniquely identifies each disposition	N/A	NOT NULL
client_id	varchar(10)	Identifies each client	N/A	NOT NULL
account_id	varchar(10)	Identifies each account	N/A	NOT NULL
type	enum	Identifies whether its user or owner	N/A	NOT NULL

Fig 12: Schema of Disposition Table

- **District:**



Name	Data type	Description	Default value	Null constraints
district_id	integer	Uniquely identifies each district	N/A	NOT NULL
city	varchar(50)	Identifies the city of the client to corresponding district	N/A	NOT NULL
state_name	varchar(50)	Identifies the state name	N/A	NOT NULL
state_abbrev	varchar(50)	Identifies State's name by short abbreviation	N/A	NOT NULL
region	varchar(50)	Identifies the region to the corresponding district	N/A	NOT NULL
division	varchar(50)	Identifies the division to the corresponding district	N/A	NOT NULL

Fig 13: Schema of District Table

C. Database Constraints

- *Primary Keys*:
  - Attribute account_id – uniquely identifies each account and is a primary key for account table
  - Attribute card_id – uniquely identifies each card and is the primary key for card table.
  - Attribute client_id – uniquely identifies each client and is a primary key for client table.
  - Attribute disp_id – uniquely identifies each disposition and is a primary key for disposition table
  - Attribute district_id – uniquely identifies each district and is a primary key for district table
  - Attribute loan_id – uniquely identifies each loan and is a primary key for loan table
  - Attribute order_id – uniquely identifies each order and is a primary key for order table
  - Attribute trans_id – uniquely identifies each transaction and is a primary key for transaction table
  - Attribute call_id – uniquely identifies each call and is a primary key for callcenterlogs table
  - Attribute complaint_id – uniquely identifies each complaint and is a primary key for crmevents table
  - Attribute review_id – uniquely identifies each review and is a primary key for crmreviews table
- *Foreign Keys*:
  - Attribute district_id – in the account table references attribute distict_id in the district table.
  - Attribute disp_id – in the card table references attribute disp_id in the disposition table.
  - Attribute district_id – in the client table references attribute district_id in the district table.
  - Attribute client_id and account_id in the disposition table references attribute client_id and account_id in the client and account tables respectively.
  - Attribute account_id – in the loan table references attribute account_id in the account table.
  - Attribute account_id – in the order table references attribute account_id in the account table.
  - Attribute account_id – in the transaction table references attribute account_id in the account table.
  - Attribute complaint_id – in the callcenterlogs table references attribute complaint_id in the crmreviews table.
  - Attribute client_id in the crmevents table references attribute client_id in the client table
  - Attribute district_id – in the crmreviews table references attribute district_id in the crmreviews table.
- *NOT NULL* and *DEFAULT*: Banking data is very prone to the information supplied by the customer and other banking officials. For this reason, no information in the tables is considered a default. This is in accordance with our use case.
  - Account - all attributes must be supplied and are not null.
  - Card - all attributes must be supplied and are not null.
  - Client - all attributes must be supplied and are not null.
  - Transaction - all attributes must be supplied and are not null.
  - order - all attributes except k_symbol must be supplied and are not null. K_symbol is the purpose of the payment and can be null.
  - loan - all attributes must be supplied and are not null.
  - call center logs - rand_client, phonefinal, vru_line, call_id and priority are null, rest have to be supplied and not null.
  - crm events - consumer complaint narrative, tags, consumerdisputed, and call_id are null, rest have to be supplied and not null.
  - crm reviews - except date and review all fields are not null.
  - disposition - all attributes must be supplied and are not null.
  - district - all attributes must be supplied and are not null.
  - *On Delete Cascades*
    - ∗ On transaction, deleting account_id deletes account(account_id).
    - ∗ On loan, deleting account_id deletes

    account(account_id).
    - On CRMEvents, deleting Client_ID deletes client(client_id).
    - On CRMCallCenterLogs, deleting Complaint ID deletes account(Complaint ID).
    - On order, deleting account_id deletes account(account_id).
    - On account, deleting district_id deletes district(district_id).
    - On card, deleting disp_id deletes disposition(disp_id).
    - On disposition, deleting client_id deletes client(client_id).
    - On disposition, deleting client_id deletes clientaccount(client_id).
    - On client, deleting district_id deletes district(district_id).
    - On CRMReviews, deleting district_id deletes district(district_id).
    - On district, deleting state_name deletes account(state_name).

V. Data Migration

The dataset contains 11 CSV files. We carefully examined each CSV file and removed the duplicate fields present in each CSV. We devised efficient data types, default values, null constraints, and on-delete constraints for each column in a relation and created the DDL scripts. We further identified the relationships between each relation and included these relations in our create.sql script. We then imported the raw CSV into the database client, trimmed the table according to our analysis using SQL queries, and achieved the resulting schema for each relation. We exported the SQL insert statement files for each relation as a separate file. To automate the process of creating the database schema and inserting the data, we made use of a shell script that internally uses the PSQL command line tool to run each .sql file over the selected database.

VI. BCNF PROVEMENT

- The relation account has the following functional dependencies account_id district_id, frequency, date. Here account_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation account is in BCNF.
- The relation card has the following functional dependencies card_id ⟶ disposition_id, type, date Here card_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation card is in BCNF
- The relation client has the following functional dependencies client_id ⟶ sex, fulldate, age, social, first, middle, last, phone, email, address_1, address_2, zipcode, district_id. Here client_id is determining all the other attributes and it is the super key.Also there is no non prime attribute that is determining other non-prime attribute(3NF) and the above functional dependency is non-trivial. Therefore,the relations client is in BCNF.

  When we tried to split the geolocation part of the client table into a separate table, we ran into issues like one zipcode being present in multiple districts and states. We found that a zip code can span multiple cities, districts, and states https://gis.stackexchange.com/questions/53918/determining-which-us-zipcodes-map-to-more-than-one-state-or-more-than-one-city. Further, the dataset documentation mentions that the zip codes are randomly generated with values between 4000 and 6000. So, we proved that the functional dependencies we came up with were invalid. Moreover, since this issue is technically valid, we left the table as is without splitting.
- The relation disposition has the following functional dependencies disp_id ⟶ client_id, account_id, type. Here disp_id is the super key, since it determines all the other attributes.But there are non-prime attributes determining another non-prime attributes (3NF fails) i.e client_id ⟶ account_id,type and disp_id. Clearly, the FD violates the BCNF. So we decompose the relation into two tables disposition(disp_id,client_id) and client_account_map(client_id,account_id,type). Both the new relations are now in BCNF.
- The relation district has following functional dependencies district_id ⟶ city, state_name, state_abbrev, region, division. state_name ⟶ state_abbrev, region, division; state_abbrev ⟶ state_name, region, division. Here in the first FD the district_id is the super key, since it determines all the other attributes. Also, the above functional dependency is non-trivial. It satisfies the BCNF conditions. But in the second and third FD's, the state_name and state_abbrev are not super keys. Therefore, the relation disposition is not in BCNF. So, it is decomposed into two relations district (district_id, city, state_name) and state(state_abbrev, state_name, region, division). Both the new relations are now in BCNF.
- The relation loan has the following functional dependencies loan_id ⟶ account_id, amount, duration, payments, status, date, location, purpose. Here loan_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation loan is in BCNF.

- The relation order has the following functional dependencies order_id ⟶ account_id, bank_to, account_to, amount, k_symbol. Here order_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation order is in BCNF
- The relation transaction has the following functional dependencies trans_id ⟶ account, amount, balance, bank string, date, type, operation. Here trans_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation transaction is in BCNF
- The relation callcenterlogs has the following functional dependencies call_id ⟶ date_received, complaint_id, rand_client, phonefinal, vru_line, priority, type, outcome, server, ser_start, ser_exit, ser_time. Here call_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation callcenterlogs is in BCNF.
- The relation crmevents has the following functional dependencies complaint_id ⟶ date received, product, sub-product, issue, sub-issue, consumer complaint narrative, tags, consumer consent provided, submitted via, date sent to company, company response to consumer, timely response, consumer disputed, client_id. Here complaint id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation crmevents is in BCNF.
- The relation crmreviews has the following functional dependencies review_id ⟶ date, stars, reviews, product, district_id. Here review_id is the super key, since it determines all the other attributes. There is no other non-prime attribute determining any other non-prime attribute(3NF). Also, the above functional dependency is non-trivial. Therefore, the relation crmreviews is in BCNF.

VII. Transformation from Initial Schema to the Final Schema

A. Final ER Diagram

After decomposing district and disposition tables using BCNF, the final Entity relationship diagram is represented in Fig. 14 It shows the relation between different tables in the database.

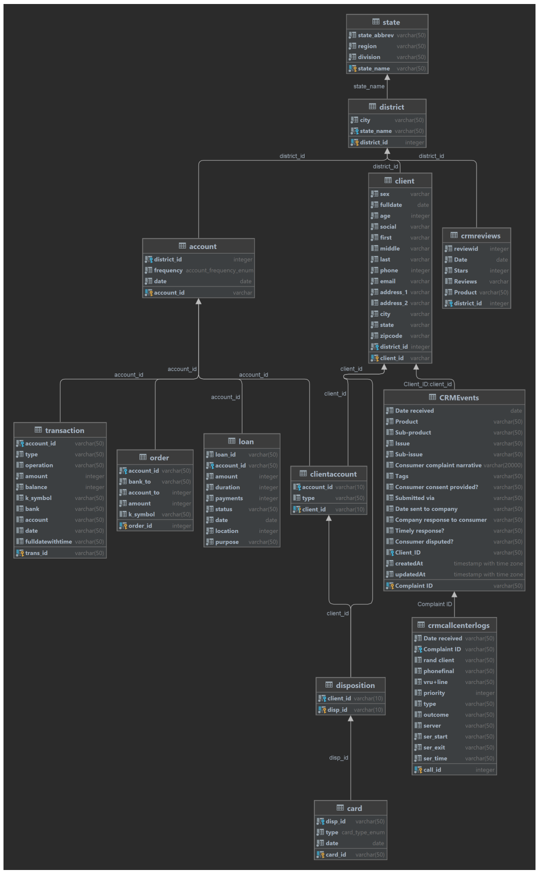
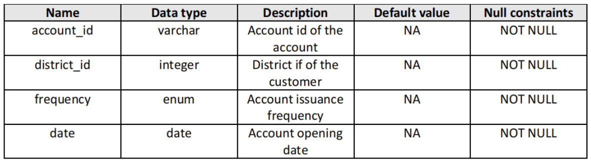


Fig 14: Final ER Diagram

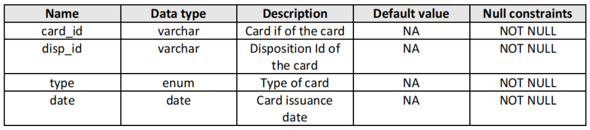
B. Relations after BCNF Provement

- We now have 13 relations in the database after BCNF decomposition.The details of the relations are described below:



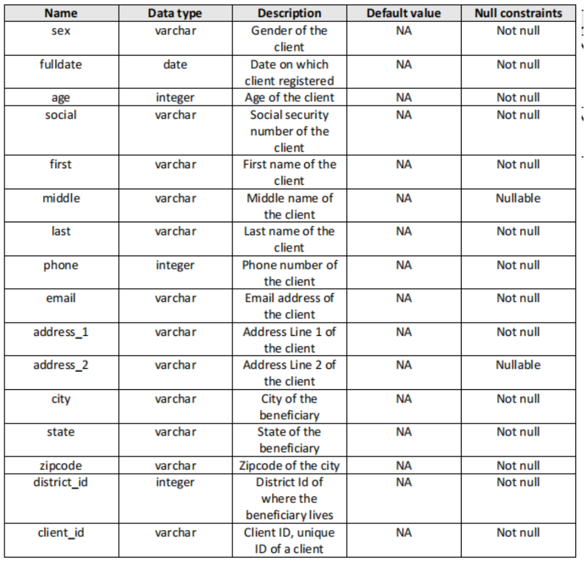
Name	Data type	Description	Default value	Null constraints
account_id	varchar	Account id of the account	NA	NOT NULL
district_id	integer	District if of the customer	NA	NOT NULL
frequency	enum	Account issuance frequency	NA	NOT NULL
date	date	Account opening date	NA	NOT NULL

Fig 15: Schema of Account table



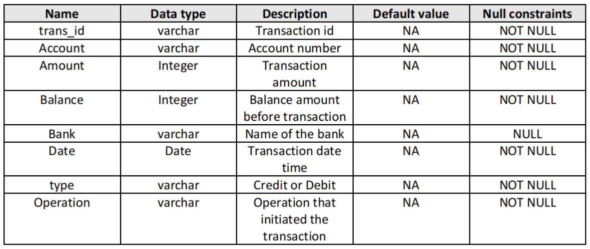
Name	Data type	Description	Default value	Null constraints
card_id	varchar	Card if of the card	NA	NOT NULL
disp_id	varchar	Disposition Id of the card	NA	NOT NULL
type	enum	Type of card	NA	NOT NULL
date	date	Card issuance date	NA	NOT NULL

Fig 16: Schema of Card Table



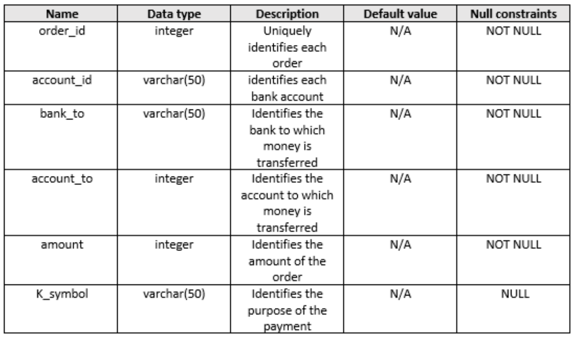
Name	Data type	Description	Default value	Null constraints
sex	varchar	Gender of the client	NA	Not null
fulldate	date	Date on which client registered	NA	Not null
age	integer	Age of the client	NA	Not null
social	varchar	Social security number of the client	NA	Not null
first	varchar	First name of the client	NA	Not null
middle	varchar	Middle name of the client	NA	Nullable
last	varchar	Last name of the client	NA	Not null
phone	integer	Phone number of the client	NA	Not null
email	varchar	Email address of the client	NA	Not null
address_1	varchar	Address Line 1 of the client	NA	Not null
address_2	varchar	Address Line 2 of the client	NA	Nullable
city	varchar	City of the beneficiary	NA	Not null
state	varchar	State of the beneficiary	NA	Not null
zipcode	varchar	Zipcode of the city	NA	Not null
district_id	integer	District Id of where the beneficiary lives	NA	Not null
client_id	varchar	Client ID, unique ID of a client	NA	Not null

Fig 17: Schema of client Table



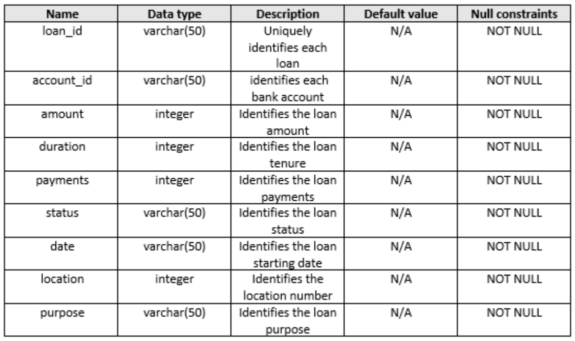
Name	Data type	Description	Default value	Null constraints
trans_id	varchar	Transaction id	NA	NOT NULL
Account	varchar	Account number	NA	NOT NULL
Amount	Integer	Transaction amount	NA	NOT NULL
Balance	Integer	Balance amount before transaction	NA	NOT NULL
Bank	varchar	Name of the bank	NA	NULL
Date	Date	Transaction date time	NA	NOT NULL
type	varchar	Credit or Debit	NA	NOT NULL
Operation	varchar	Operation that initiated the transaction	NA	NOT NULL

Fig 18: Schema of transaction table



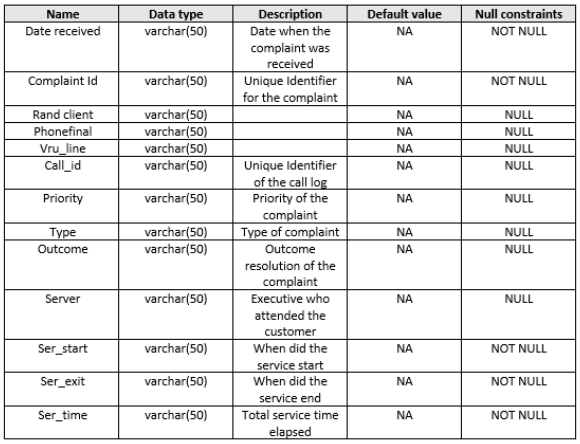
Name	Data type	Description	Default value	Null constraints
order_id	integer	Uniquely identifies each order	N/A	NOT NULL
account_id	varchar(50)	identifies each bank account	N/A	NOT NULL
bank_to	varchar(50)	Identifies the bank to which money is transferred	N/A	NOT NULL
account_to	integer	Identifies the account to which money is transferred	N/A	NOT NULL
amount	integer	Identifies the amount of the order	N/A	NOT NULL
K_symbol	varchar(50)	Identifies the purpose of the payment	N/A	NULL

Fig 19: Schema of order Table



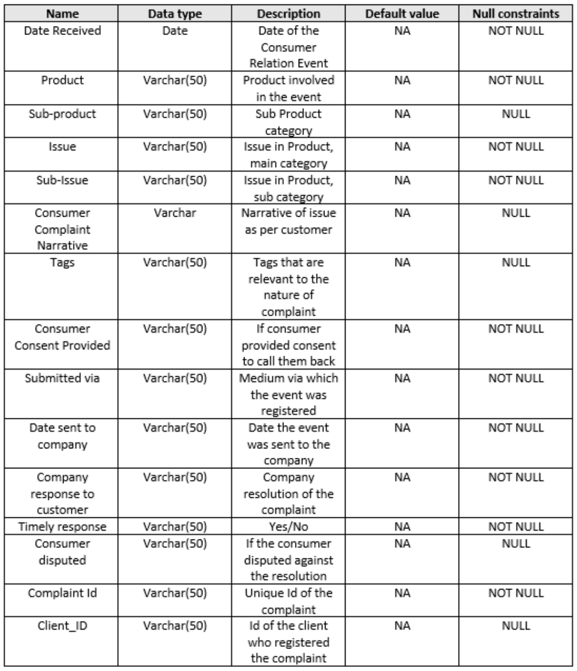
Name	Data type	Description	Default value	Null constraints
loan_id	varchar(50)	Uniquely identifies each loan	N/A	NOT NULL
account_id	varchar(50)	identifies each bank account	N/A	NOT NULL
amount	integer	Identifies the loan amount	N/A	NOT NULL
duration	integer	Identifies the loan tenure	N/A	NOT NULL
payments	integer	Identifies the loan payments	N/A	NOT NULL
status	varchar(50)	Identifies the loan status	N/A	NOT NULL
date	varchar(50)	Identifies the loan starting date	N/A	NOT NULL
location	integer	Identifies the location number	N/A	NOT NULL
purpose	varchar(50)	Identifies the loan purpose	N/A	NOT NULL

Fig 20: Schema of loan table



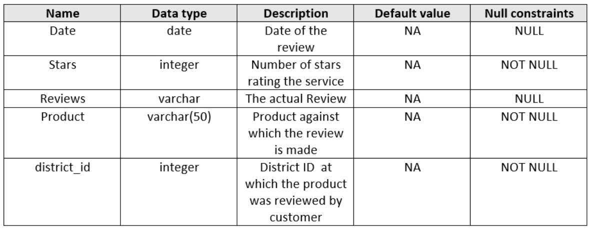
Name	Data type	Description	Default value	Null constraints
Date received	varchar(50)	Date when the complaint was received	NA	NOT NULL
Complaint Id	varchar(50)	Unique Identifier for the complaint	NA	NOT NULL
Rand client	varchar(50)		NA	NULL
Phonefinal	varchar(50)		NA	NULL
Vru_line	varchar(50)		NA	NULL
Call_id	varchar(50)	Unique Identifier of the call log	NA	NULL
Priority	varchar(50)	Priority of the complaint	NA	NULL
Type	varchar(50)	Type of complaint	NA	NULL
Outcome	varchar(50)	Outcome resolution of the complaint	NA	NULL
Server	varchar(50)	Executive who attended the customer	NA	NULL
Ser_start	varchar(50)	When did the service start	NA	NOT NULL
Ser_exit	varchar(50)	When did the service end	NA	NOT NULL
Ser_time	varchar(50)	Total service time elapsed	NA	NOT NULL

Fig 21: Schema of Call center logs Table



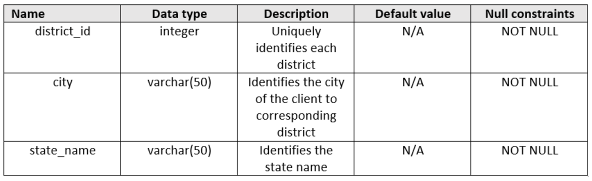
Name	Data type	Description	Default value	Null constraints
Date Received	Date	Date of the Consumer Relation Event	NA	NOT NULL
Product	Varchar(50)	Product involved in the event	NA	NOT NULL
Sub-product	Varchar(50)	Sub Product category	NA	NULL
Issue	Varchar(50)	Issue in Product, main category	NA	NOT NULL
Sub-Issue	Varchar(50)	Issue in Product, sub category	NA	NOT NULL
Consumer Complaint Narrative	Varchar	Narrative of issue as per customer	NA	NULL
Tags	Varchar(50)	Tags that are relevant to the nature of complaint	NA	NULL
Consumer Consent Provided	Varchar(50)	If consumer provided consent to call them back	NA	NOT NULL
Submitted via	Varchar(50)	Medium via which the event was registered	NA	NOT NULL
Date sent to company	Varchar(50)	Date the event was sent to the company	NA	NOT NULL
Company response to customer	Varchar(50)	Company resolution of the complaint	NA	NOT NULL
Timely response	Varchar(50)	Yes/No	NA	NOT NULL
Consumer disputed	Varchar(50)	If the consumer disputed against the resolution	NA	NULL
Complaint Id	Varchar(50)	Unique Id of the complaint	NA	NOT NULL
Client_ID	Varchar(50)	Id of the client who registered the complaint	NA	NULL

Fig 22: Schema of CRM Events Table



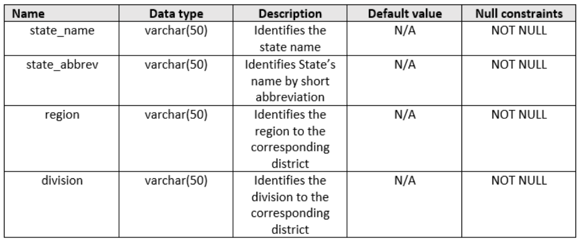
Name	Data type	Description	Default value	Null constraints
Date	date	Date of the review	NA	NULL
Stars	integer	Number of stars rating the service	NA	NOT NULL
Reviews	varchar	The actual Review	NA	NULL
Product	varchar(50)	Product against which the review is made	NA	NOT NULL
district_id	integer	District ID at which the product was reviewed by customer	NA	NOT NULL

Fig 23: Schema of CRM Reviews Table



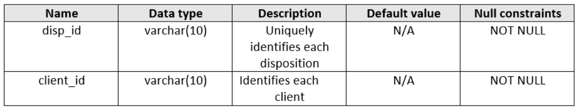
Name	Data type	Description	Default value	Null constraints
district_id	integer	Uniquely identifies each district	N/A	NOT NULL
city	varchar(50)	Identifies the city of the client to corresponding district	N/A	NOT NULL
state_name	varchar(50)	Identifies the state name	N/A	NOT NULL

Fig 24: Schema of District Table



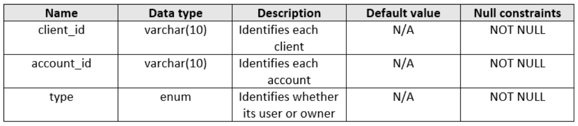
Name	Data type	Description	Default value	Null constraints
state_name	varchar(50)	Identifies the state name	N/A	NOT NULL
state_abbrev	varchar(50)	Identifies State's name by short abbreviation	N/A	NOT NULL
region	varchar(50)	Identifies the region to the corresponding district	N/A	NOT NULL
division	varchar(50)	Identifies the division to the corresponding district	N/A	NOT NULL

Fig 25: Schema of State Table



Name	Data type	Description	Default value	Null constraints
disp_id	varchar(10)	Uniquely identifies each disposition	N/A	NOT NULL
client_id	varchar(10)	Identifies each client	N/A	NOT NULL

Fig 26: Schema of Disposition Table



Name	Data type	Description	Default value	Null constraints
client_id	varchar(10)	Identifies each client	N/A	NOT NULL
account_id	varchar(10)	Identifies each account	N/A	NOT NULL
type	enum	Identifies whether its user or owner	N/A	NOT NULL

Fig 27: Schema of Client Account Table

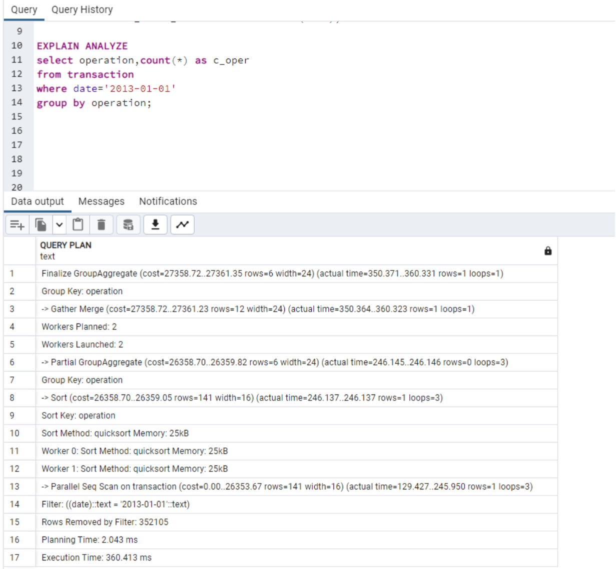
C. *Database Constraints for tables formed after decomposition*

- *Primary Keys*:
  - ∗ Attribute state_name – uniquely identifies each state and is a primary key for state table
  - ∗ Attribute client_id and account_id – uniquely identifies each client account and is a composite primary key for clientaccount table
  - ∗ Attribute disp_id – uniquely identifies each disposition and is a primary key for the disposition table
  - ∗ Attribute disrict_id – uniquely identifies each district and is a primary key for the district table
- *Nullable and Default Constraints*:

  for all the tables there is no default constraint and all the attributes are non-nullable.

VIII. Problems while handling larger data sets

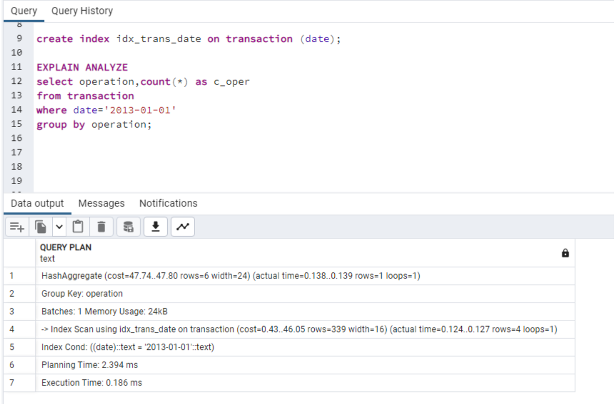
One of the problems that we have observed using larger data sets is taking longer execution times.one such scenario is when we want to query transaction table on date columns. Since the transaction table is huge, it is taking longer time to retrieve data. So, we have adopted indexing concept on the date column and have observed the execution time reduced drastically from 360.413 ms to 0.186 ms.



```
EXPLAIN ANALYZE
select operation,count(*) as c_oper
from transaction
where date='2013-01-01'
group by operation;
```

	QUERY PLAN text
1	Finalize GroupAggregate (cost=27358.72..27361.35 rows=6 width=24) (actual time=350.371..360.331 rows=1 loops=1)
2	Group Key: operation
3	-> Gather Merge (cost=27358.72..27361.23 rows=12 width=24) (actual time=350.364..360.323 rows=1 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Partial GroupAggregate (cost=26358.70..26359.82 rows=6 width=24) (actual time=246.145..246.146 rows=0 loops=3)
7	Group Key: operation
8	-> Sort (cost=26358.70..26359.05 rows=141 width=16) (actual time=246.137..246.137 rows=1 loops=3)
9	Sort Key: operation
10	Sort Method: quicksort Memory: 25kB
11	Worker 0: Sort Method: quicksort Memory: 25kB
12	Worker 1: Sort Method: quicksort Memory: 25kB
13	-> Parallel Seq Scan on transaction (cost=0.00..26353.67 rows=141 width=16) (actual time=129.427..245.950 rows=1 loops=3)
14	Filter: ((date)::text = '2013-01-01'::text)
15	Rows Removed by Filter: 352105
16	Planning Time: 2.043 ms
17	Execution Time: 360.413 ms

Fig 25: Before - Without Indexing



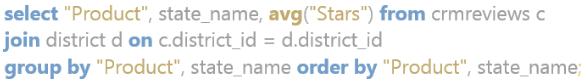
```
create index idx_trans_date on transaction (date);

EXPLAIN ANALYZE
select operation,count(*) as c_oper
from transaction
where date='2013-01-01'
group by operation;
```

	QUERY PLAN text
1	HashAggregate (cost=47.74..47.80 rows=6 width=24) (actual time=0.138..0.139 rows=1 loops=1)
2	Group Key: operation
3	Batches: 1 Memory Usage: 24kB
4	-> Index Scan using idx_trans_date on transaction (cost=0.43..46.05 rows=339 width=16) (actual time=0.124..0.127 rows=4 loops=1)
5	Index Cond: ((date)::text = '2013-01-01'::text)
6	Planning Time: 2.394 ms
7	Execution Time: 0.186 ms

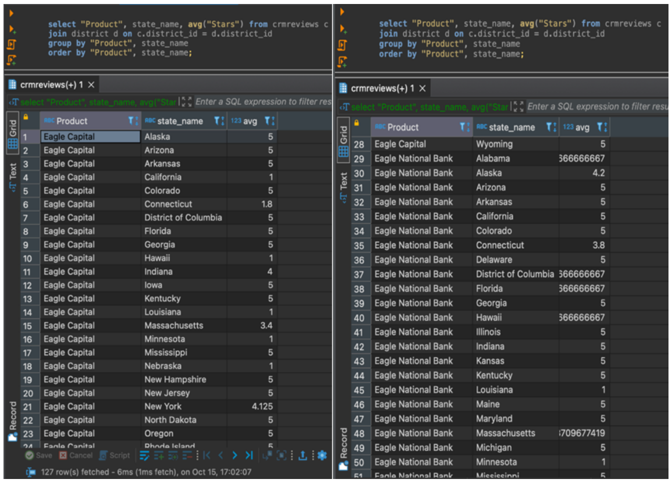
Fig 26: After - Indexing on date

IX. Basic Queries

1) **Find the average rating of each product of Eagle National Bank for each state**



```
select "Product", state_name, avg("Stars") from crmreviews c
join district d on c.district_id = d.district_id
group by "Product", state_name order by "Product", state_name;
```



```
select "Product", state_name, avg("Stars") from crmreviews c
join district d on c.district_id = d.district_id
group by "Product", state_name
order by "Product", state_name;
```

crmreviews(+) 1

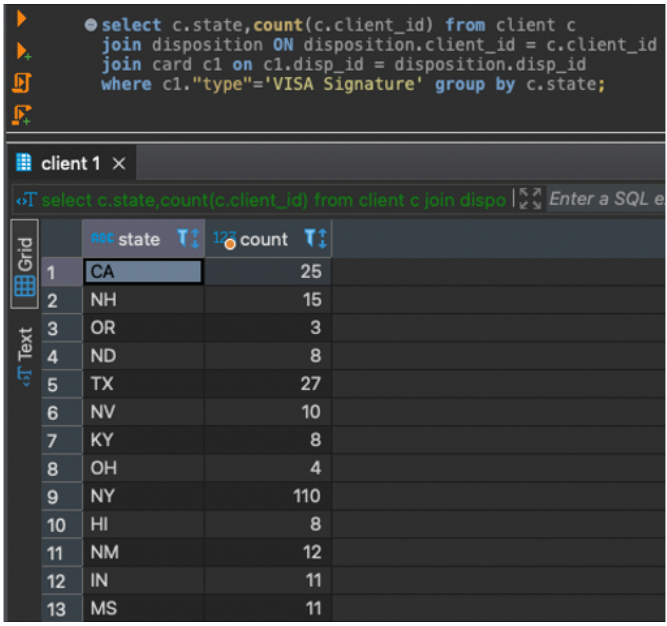
	Product	state_name	avg
1	Eagle Capital	Alaska	5
2	Eagle Capital	Arizona	5
3	Eagle Capital	Arkansas	5
4	Eagle Capital	California	1
5	Eagle Capital	Colorado	5
6	Eagle Capital	Connecticut	1.8
7	Eagle Capital	District of Columbia	5
8	Eagle Capital	Florida	5
9	Eagle Capital	Georgia	5
10	Eagle Capital	Hawaii	1
11	Eagle Capital	Indiana	4
12	Eagle Capital	Iowa	5
13	Eagle Capital	Kentucky	5
14	Eagle Capital	Louisiana	1
15	Eagle Capital	Massachusetts	3.4
16	Eagle Capital	Minnesota	1
17	Eagle Capital	Mississippi	5
18	Eagle Capital	Nebraska	1
19	Eagle Capital	New Hampshire	5
20	Eagle Capital	New Jersey	5
21	Eagle Capital	New York	4.125
22	Eagle Capital	North Dakota	5
23	Eagle Capital	Oregon	5
24	Eagle Capital	Rhode Island	5

127 row(s) fetched - 6ms (1ms fetch), on Oct 15, 17:02:07

	Product	state_name	avg
28	Eagle Capital	Wyoming	5
29	Eagle National Bank	Alabama	366666667
30	Eagle National Bank	Alaska	4.2
31	Eagle National Bank	Arizona	5
32	Eagle National Bank	Arkansas	5
33	Eagle National Bank	California	5
34	Eagle National Bank	Colorado	5
35	Eagle National Bank	Connecticut	3.8
36	Eagle National Bank	Delaware	5
37	Eagle National Bank	District of Columbia	366666667
38	Eagle National Bank	Florida	366666667
39	Eagle National Bank	Georgia	5
40	Eagle National Bank	Hawaii	366666667
41	Eagle National Bank	Illinois	5
42	Eagle National Bank	Indiana	5
43	Eagle National Bank	Kansas	5
44	Eagle National Bank	Kentucky	5
45	Eagle National Bank	Louisiana	1
46	Eagle National Bank	Maine	5
47	Eagle National Bank	Maryland	5
48	Eagle National Bank	Massachusetts	709677419
49	Eagle National Bank	Michigan	5
50	Eagle National Bank	Minnesota	1
51	Eagle National Bank	Mississippi	5

2) **Find the distribution of clients in each state that has VISA signature Card.**

```
select c.state,count(c.client_id) from client c
join disposition ON disposition.client_id = c.client_id
join card c1 on c1.disp_id = disposition.disp_id
where c1."type"='VISA Signature' group by c.state;
```



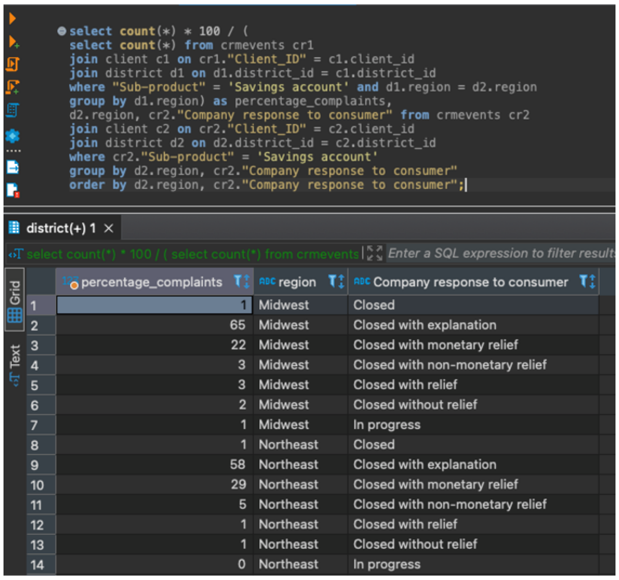
```
select c.state,count(c.client_id) from client c
join disposition ON disposition.client_id = c.client_id
join card c1 on c1.disp_id = disposition.disp_id
where c1."type"='VISA Signature' group by c.state;
```

client 1

	state	count
1	CA	25
2	NH	15
3	OR	3
4	ND	8
5	TX	27
6	NV	10
7	KY	8
8	OH	4
9	NY	110
10	HI	8
11	NM	12
12	IN	11
13	MS	11

3) **For a particular sub-product, let's say "Savings Account", find how the customer support performed in terms of how many queries by customers were closed/open per region.**

```
select count(*) * 100 / (
select count(*) from crmevents cr1
join client c1 on cr1."Client_ID" = c1.client_id
join district d1 on d1.district_id = c1.district_id
where "Sub-product" = 'Savings account' and d1.region = d2.region
group by d1.region) as percentage_complaints,
d2.region, cr2."Company response to consumer" from crmevents cr2
join client c2 on cr2."Client_ID" = c2.client_id
join district d2 on d2.district_id = c2.district_id
where cr2."Sub-product" = 'Savings account'
group by d2.region, cr2."Company response to consumer"
order by d2.region, cr2."Company response to consumer";
```



```
select count(*) * 100 / (
select count(*) from crmevents cr1
join client c1 on cr1."Client_ID" = c1.client_id
join district d1 on d1.district_id = c1.district_id
where "Sub-product" = 'Savings account' and d1.region = d2.region
group by d1.region) as percentage_complaints,
d2.region, cr2."Company response to consumer" from crmevents cr2
join client c2 on cr2."Client_ID" = c2.client_id
join district d2 on d2.district_id = c2.district_id
where cr2."Sub-product" = 'Savings account'
group by d2.region, cr2."Company response to consumer"
order by d2.region, cr2."Company response to consumer";
```

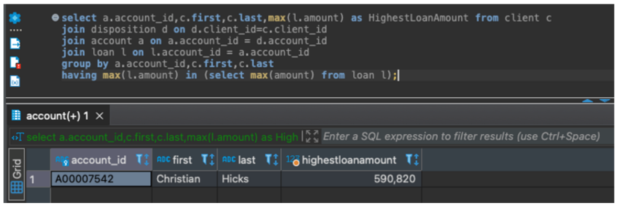
district(+) 1

	percentage_complaints	region	Company response to consumer
1	1	Midwest	Closed
2	65	Midwest	Closed with explanation
3	22	Midwest	Closed with monetary relief
4	3	Midwest	Closed with non-monetary relief
5	3	Midwest	Closed with relief
6	2	Midwest	Closed without relief
7	1	Midwest	In progress
8	1	Northeast	Closed
9	58	Northeast	Closed with explanation
10	29	Northeast	Closed with monetary relief
11	5	Northeast	Closed with non-monetary relief
12	1	Northeast	Closed with relief
13	1	Northeast	Closed without relief
14	0	Northeast	In progress

Grid
Text
select count(*) * 100 / (select count(*) from crmevents
Enter a SQL expression to filter result

4) **Find the account details of the person that has taken the highest loan amount**

```
select a.account_id,c.first,c.last,max(l.amount) as HighestLoanAmount from
client c
join disposition d on d.client_id=c.client_id
join account a on a.account_id = d.account_id
join loan l on l.account_id = a.account_id
group by a.account_id,c.first,c.last
having max(l.amount) in (select max(amount) from loan l);
```



```
select a.account_id,c.first,c.last,max(l.amount) as HighestLoanAmount from client c
join disposition d on d.client_id=c.client_id
join account a on a.account_id = d.account_id
join loan l on l.account_id = a.account_id
group by a.account_id,c.first,c.last
having max(l.amount) in (select max(amount) from loan l);
```

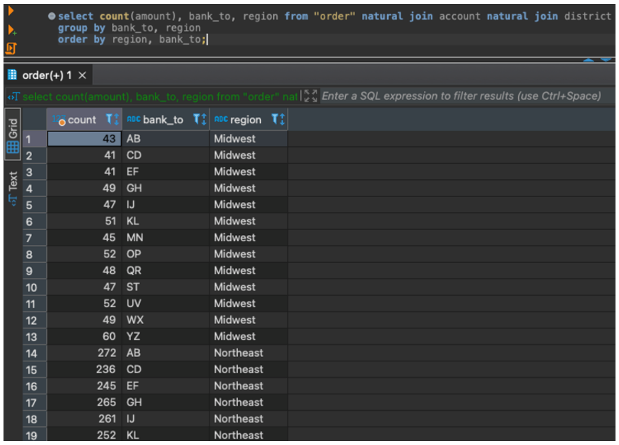
account(+) 1

	account_id	first	last	highestloanamount
1	A00007542	Christian	Hicks	590,820

Grid
select a.account_id,c.first,c.last,max(l.amount) as High
Enter a SQL expression to filter results (use Ctrl+Space)

5) **In any particular region, how much money was transferred from Eagle National Bank to other banks?**

```
select count(amount), bank_to, region from "order"
natural join account
natural join district
group by bank_to, region
order by region, bank_to;
```

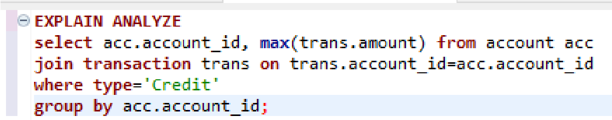


```
select count(amount), bank_to, region from "order" natural join account natural join district
group by bank_to, region
order by region, bank_to;
```

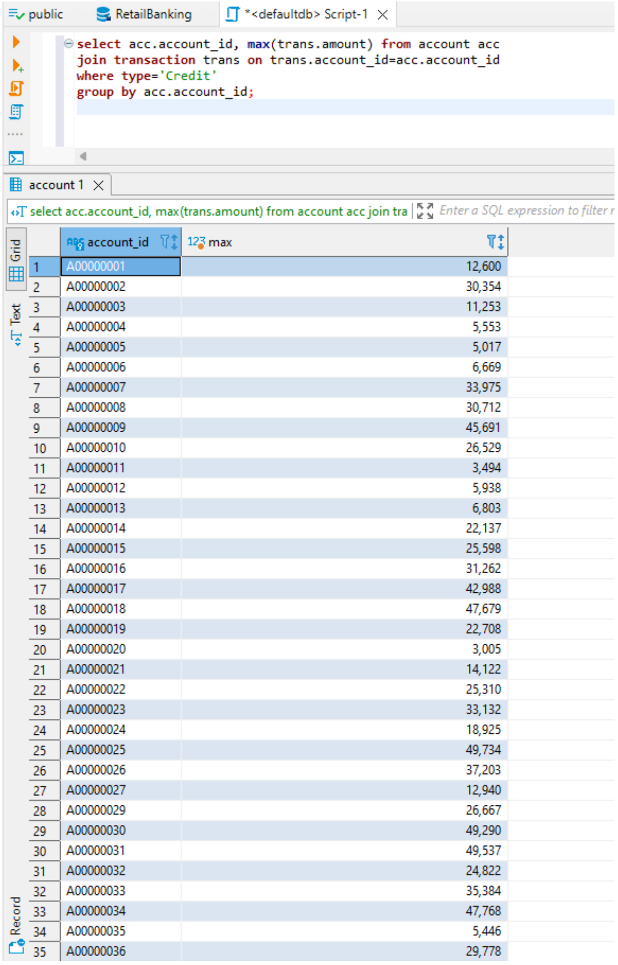
order(+) 1
select count(amount), bank_to, region from "order" nat
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text

	count	bank_to	region
1	43	AB	Midwest
2	41	CD	Midwest
3	41	EF	Midwest
4	49	GH	Midwest
5	47	IJ	Midwest
6	51	KL	Midwest
7	45	MN	Midwest
8	52	OP	Midwest
9	48	QR	Midwest
10	47	ST	Midwest
11	52	UV	Midwest
12	49	WX	Midwest
13	60	YZ	Midwest
14	272	AB	Northeast
15	236	CD	Northeast
16	245	EF	Northeast
17	265	GH	Northeast
18	261	IJ	Northeast
19	252	KL	Northeast

6) **Find the maximum transaction of each account using Credit as transaction type**



```
EXPLAIN ANALYZE
select acc.account_id, max(trans.amount) from account acc
join transaction trans on trans.account_id=acc.account_id
where type='Credit'
group by acc.account_id;
```

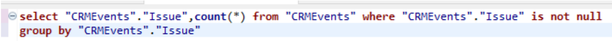


public
RetailBanking
*<defaultdb> Script-1
account 1
select acc.account_id, max(trans.amount) from account acc join tra
Enter a SQL expression to filter r
Grid
Text
Record

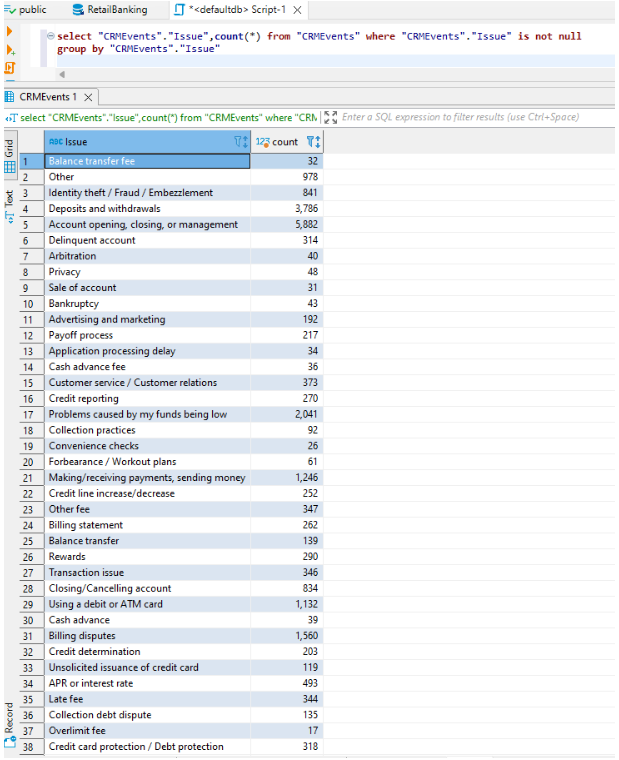
```
select acc.account_id, max(trans.amount) from account acc
join transaction trans on trans.account_id=acc.account_id
where type='Credit'
group by acc.account_id;
```

	account_id	max
1	A00000001	12,600
2	A00000002	30,354
3	A00000003	11,253
4	A00000004	5,553
5	A00000005	5,017
6	A00000006	6,669
7	A00000007	33,975
8	A00000008	30,712
9	A00000009	45,691
10	A00000010	26,529
11	A00000011	3,494
12	A00000012	5,938
13	A00000013	6,803
14	A00000014	22,137
15	A00000015	25,598
16	A00000016	31,262
17	A00000017	42,988
18	A00000018	47,679
19	A00000019	22,708
20	A00000020	3,005
21	A00000021	14,122
22	A00000022	25,310
23	A00000023	33,132
24	A00000024	18,925
25	A00000025	49,734
26	A00000026	37,203
27	A00000027	12,940
28	A00000029	26,667
29	A00000030	49,290
30	A00000031	49,537
31	A00000032	24,822
32	A00000033	35,384
33	A00000034	47,768
34	A00000035	5,446
35	A00000036	29,778

7) **Find the count of number of events registered for each Issue in crmevents which are not null**



```
select "CRMEvents"."Issue",count(*) from "CRMEvents" where "CRMEvents"."Issue" is not null
group by "CRMEvents"."Issue"
```

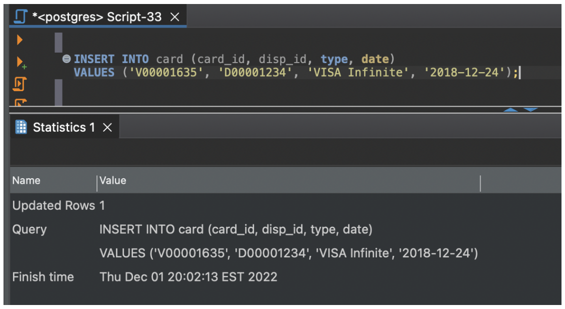


public
RetailBanking
*<defaultdb> Script-1
CRMEvents 1
select "CRMEvents"."Issue",count(*) from "CRMEvents" where "CRM
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record

```
select "CRMEvents"."Issue",count(*) from "CRMEvents" where "CRMEvents"."Issue" is not null
group by "CRMEvents"."Issue"
```

	Issue	count
1	Balance transfer fee	32
2	Other	978
3	Identity theft / Fraud / Embezzlement	841
4	Deposits and withdrawals	3,786
5	Account opening, closing, or management	5,882
6	Delinquent account	314
7	Arbitration	40
8	Privacy	48
9	Sale of account	31
10	Bankruptcy	43
11	Advertising and marketing	192
12	Payoff process	217
13	Application processing delay	34
14	Cash advance fee	36
15	Customer service / Customer relations	373
16	Credit reporting	270
17	Problems caused by my funds being low	2,041
18	Collection practices	92
19	Convenience checks	26
20	Forbearance / Workout plans	61
21	Making/receiving payments, sending money	1,246
22	Credit line increase/decrease	252
23	Other fee	347
24	Billing statement	262
25	Balance transfer	139
26	Rewards	290
27	Transaction issue	346
28	Closing/Cancelling account	834
29	Using a debit or ATM card	1,132
30	Cash advance	39
31	Billing disputes	1,560
32	Credit determination	203
33	Unsolicited issuance of credit card	119
34	APR or interest rate	493
35	Late fee	344
36	Collection debt dispute	135
37	Overlimit fee	17
38	Credit card protection / Debt protection	318

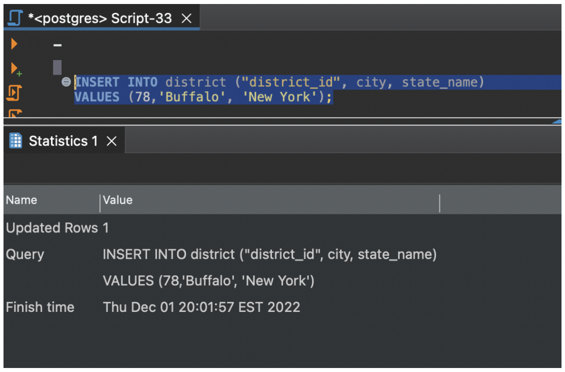
1) Insert / Update Queries:



```
INSERT INTO card (card_id, disp_id, type, date)
VALUES ('V00001635', 'D00001234', 'VISA Infinite', '2018-12-24');
```

```
Name          Value
Updated Rows  1
Query         INSERT INTO card (card_id, disp_id, type, date)
              VALUES ('V00001635', 'D00001234', 'VISA Infinite', '2018-12-24')
Finish time   Thu Dec 01 20:02:13 EST 2022
```

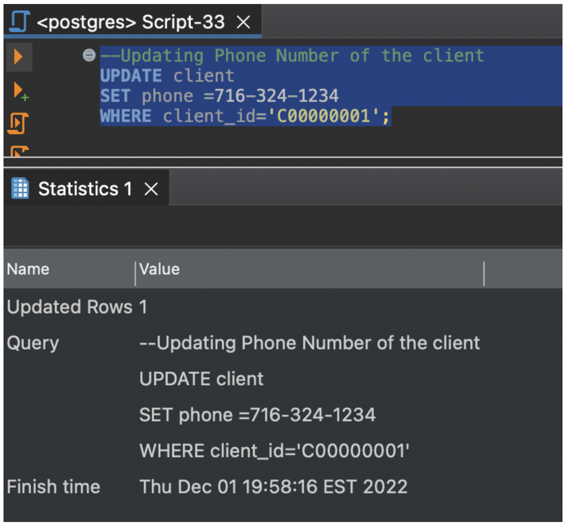
Fig 36: Insert Card details



```
INSERT INTO district ("district_id", city, state_name)
VALUES (78,'Buffalo', 'New York');
```

```
Name          Value
Updated Rows  1
Query         INSERT INTO district ("district_id", city, state_name)
              VALUES (78,'Buffalo', 'New York')
Finish time   Thu Dec 01 20:01:57 EST 2022
```

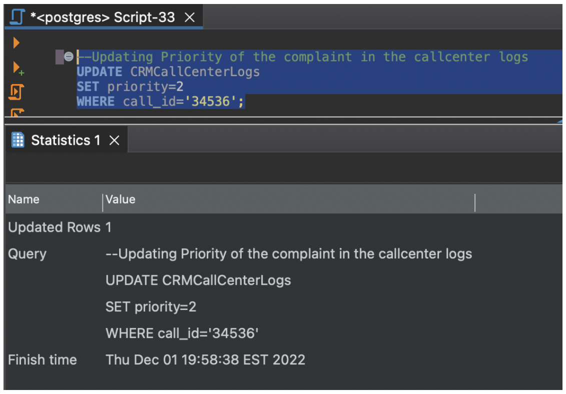
Fig 37: Add a district



```
--Updating Phone Number of the client
UPDATE client
SET phone =716-324-1234
WHERE client_id='C00000001';
```

```
Name           Value
Updated Rows   1
Query          --Updating Phone Number of the client
               UPDATE client
               SET phone =716-324-1234
               WHERE client_id='C00000001'
Finish time    Thu Dec 01 19:58:16 EST 2022
```

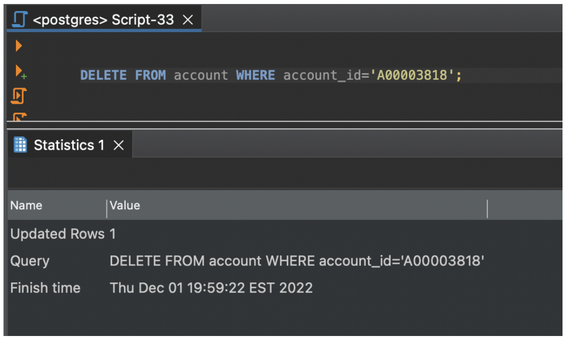
Fig 38: Update phone number of a client



```
--Updating Priority of the complaint in the callcenter logs
UPDATE CRMCallCenterLogs
SET priority=2
WHERE call_id='34536';
```

```
Name           Value
Updated Rows   1
Query          --Updating Priority of the complaint in the callcenter logs
               UPDATE CRMCallCenterLogs
               SET priority=2
               WHERE call_id='34536'
Finish time    Thu Dec 01 19:58:38 EST 2022
```

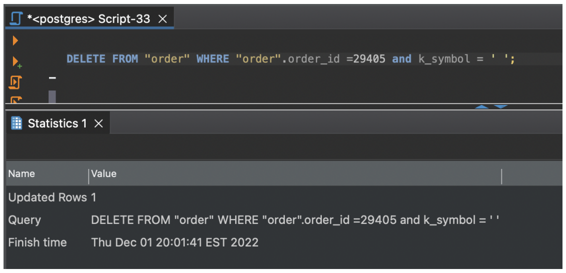
Fig 39: Update priority of a complaint



```
DELETE FROM account WHERE account_id='A00003818';
```

```
Name           Value
Updated Rows   1
Query          DELETE FROM account WHERE account_id='A00003818'
Finish time    Thu Dec 01 19:59:22 EST 2022
```

Fig 40: delete account based on account_id



```
DELETE FROM "order" WHERE "order".order_id =29405 and k_symbol = ' ';
```

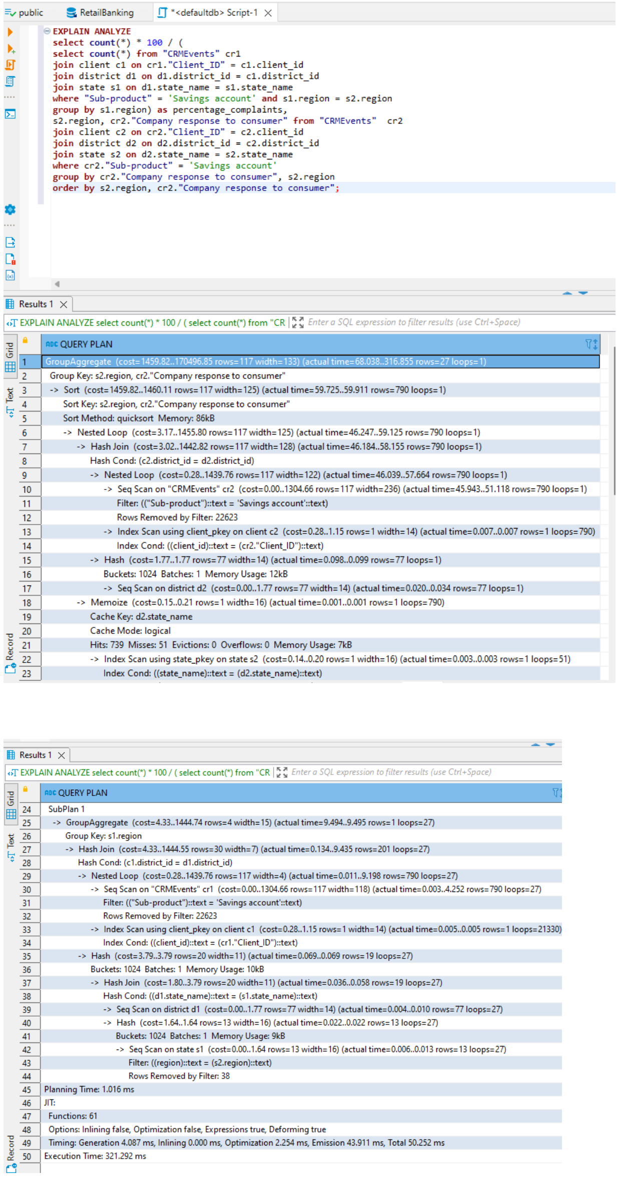
```
Name           Value
Updated Rows   1
Query          DELETE FROM "order" WHERE "order".order_id =29405 and k_symbol = ' '
Finish time    Thu Dec 01 20:01:41 EST 2022
```

Fig 41: Delete order with order id and empty K_symbol

X. QUERY EXECUTION ANALYSIS

In this part, three of the queries has been executed and analyzed using 'EXPLAIN ANALYZE'.

1) **For a particular sub-product, let's say "Savings Account", find how the customer support performed in terms of how many queries by customers were closed/open per region.**

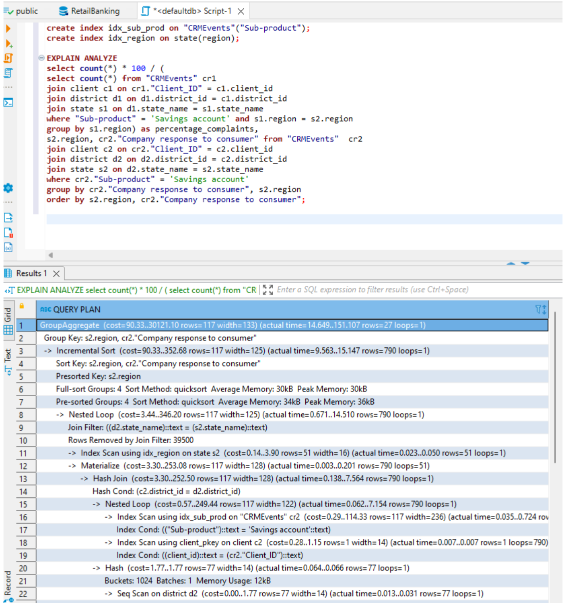


```
EXPLAIN ANALYZE
select count(*) * 100 / (
select count(*) from "CRMEvents" cr1
join client c1 on cr1."Client_ID" = c1.client_id
join district d1 on d1.district_id = c1.district_id
join state s1 on d1.state_name = s1.state_name
where "Sub-product" = 'Savings account' and s1.region = s2.region
group by s1.region) as percentage_complaints,
s2.region, cr2."Company response to consumer" from "CRMEvents"  cr2
join client c2 on cr2."Client_ID" = c2.client_id
join district d2 on d2.district_id = c2.district_id
join state s2 on d2.state_name = s2.state_name
where cr2."Sub-product" = 'Savings account'
group by cr2."Company response to consumer", s2.region
order by s2.region, cr2."Company response to consumer";
```

```
QUERY PLAN
GroupAggregate  (cost=1459.82..170496.85 rows=117 width=133) (actual time=68.038..316.855 rows=27 loops=1)
  Group Key: s2.region, cr2."Company response to consumer"
  ->  Sort  (cost=1459.82..1460.11 rows=117 width=125) (actual time=59.725..59.911 rows=790 loops=1)
        Sort Key: s2.region, cr2."Company response to consumer"
        Sort Method: quicksort  Memory: 86kB
        ->  Nested Loop  (cost=3.17..1455.80 rows=117 width=125) (actual time=46.247..59.125 rows=790 loops=1)
              ->  Hash Join  (cost=3.02..1442.82 rows=117 width=128) (actual time=46.184..58.155 rows=790 loops=1)
                    Hash Cond: (c2.district_id = d2.district_id)
                    ->  Nested Loop  (cost=0.28..1439.76 rows=117 width=122) (actual time=46.039..57.664 rows=790 loops=1)
                          ->  Seq Scan on "CRMEvents" cr2  (cost=0.00..1304.66 rows=117 width=236) (actual time=45.943..51.118 rows=790 loops=1)
                                Filter: (("Sub-product")::text = 'Savings account'::text)
                                Rows Removed by Filter: 22623
                          ->  Index Scan using client_pkey on client c2  (cost=0.28..1.15 rows=1 width=14) (actual time=0.007..0.007 rows=1 loops=790)
                                Index Cond: ((client_id)::text = (cr2."Client_ID")::text)
                    ->  Hash  (cost=1.77..1.77 rows=77 width=14) (actual time=0.098..0.099 rows=77 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 12kB
                          ->  Seq Scan on district d2  (cost=0.00..1.77 rows=77 width=14) (actual time=0.020..0.034 rows=77 loops=1)
              ->  Memoize  (cost=0.15..0.21 rows=1 width=16) (actual time=0.001..0.001 rows=1 loops=790)
                    Cache Key: d2.state_name
                    Cache Mode: logical
                    Hits: 739  Misses: 51  Evictions: 0  Overflows: 0  Memory Usage: 7kB
                    ->  Index Scan using state_pkey on state s2  (cost=0.14..0.20 rows=1 width=16) (actual time=0.003..0.003 rows=1 loops=51)
                          Index Cond: ((state_name)::text = (d2.state_name)::text)
  SubPlan 1
    ->  GroupAggregate  (cost=4.33..1444.74 rows=4 width=15) (actual time=9.494..9.495 rows=1 loops=27)
          Group Key: s1.region
          ->  Hash Join  (cost=4.33..1444.55 rows=30 width=7) (actual time=0.134..9.435 rows=201 loops=27)
                Hash Cond: (c1.district_id = d1.district_id)
                ->  Nested Loop  (cost=0.28..1439.76 rows=117 width=4) (actual time=0.011..9.198 rows=790 loops=27)
                      ->  Seq Scan on "CRMEvents" cr1  (cost=0.00..1304.66 rows=117 width=118) (actual time=0.003..4.252 rows=790 loops=27)
                            Filter: (("Sub-product")::text = 'Savings account'::text)
                            Rows Removed by Filter: 22623
                      ->  Index Scan using client_pkey on client c1  (cost=0.28..1.15 rows=1 width=14) (actual time=0.005..0.005 rows=1 loops=21330)
                            Index Cond: ((client_id)::text = (cr1."Client_ID")::text)
                ->  Hash  (cost=3.79..3.79 rows=20 width=11) (actual time=0.069..0.069 rows=19 loops=27)
                      Buckets: 1024  Batches: 1  Memory Usage: 10kB
                      ->  Hash Join  (cost=1.80..3.79 rows=20 width=11) (actual time=0.036..0.058 rows=19 loops=27)
                            Hash Cond: ((d1.state_name)::text = (s1.state_name)::text)
                            ->  Seq Scan on district d1  (cost=0.00..1.77 rows=77 width=14) (actual time=0.004..0.010 rows=77 loops=27)
                            ->  Hash  (cost=1.64..1.64 rows=13 width=16) (actual time=0.022..0.022 rows=13 loops=27)
                                  Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                  ->  Seq Scan on state s1  (cost=0.00..1.64 rows=13 width=16) (actual time=0.006..0.013 rows=13 loops=27)
                                        Filter: ((region)::text = (s2.region)::text)
                                        Rows Removed by Filter: 38
Planning Time: 1.016 ms
JIT:
  Functions: 61
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 4.087 ms, Inlining 0.000 ms, Optimization 2.254 ms, Emission 43.911 ms, Total 50.252 ms
Execution Time: 321.292 ms
```

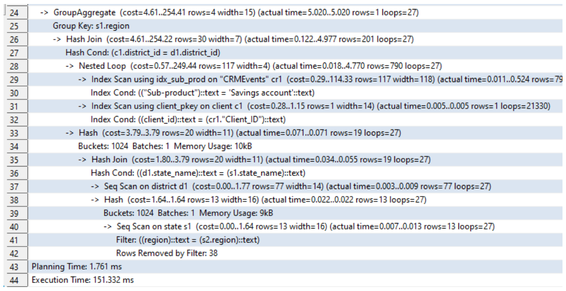
Fig 27: Before - Without Indexing

Here cost=1459.82 and Execution Time: 321.292 ms Lets now index on Sub-product column in CRMEvents table and region column on state as they are executing in sequential scan which is making the query execution time and cost more.


public
RetailBanking
*<defaultdb> Script-1
Results 1
EXPLAIN ANALYZE select count(*) * 100 / (select count(*) from "CR
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text
Record

```
create index idx_sub_prod on "CRMEvents"("Sub-product");
create index idx_region on state(region);

EXPLAIN ANALYZE
select count(*) * 100 / (
select count(*) from "CRMEvents" cr1
join client c1 on cr1."Client_ID" = c1.client_id
join district d1 on d1.district_id = c1.district_id
join state s1 on d1.state_name = s1.state_name
where "Sub-product" = 'Savings account' and s1.region = s2.region
group by s1.region) as percentage_complaints,
s2.region, cr2."Company response to consumer" from "CRMEvents"  cr2
join client c2 on cr2."Client_ID" = c2.client_id
join district d2 on d2.district_id = c2.district_id
join state s2 on d2.state_name = s2.state_name
where cr2."Sub-product" = 'Savings account'
group by cr2."Company response to consumer", s2.region
order by s2.region, cr2."Company response to consumer";
```

```
QUERY PLAN
GroupAggregate  (cost=90.33..30121.10 rows=117 width=133) (actual time=14.649..151.107 rows=27 loops=1)
  Group Key: s2.region, cr2."Company response to consumer"
  ->  Incremental Sort  (cost=90.33..352.68 rows=117 width=125) (actual time=9.563..15.147 rows=790 loops=1)
        Sort Key: s2.region, cr2."Company response to consumer"
        Presorted Key: s2.region
        Full-sort Groups: 4  Sort Method: quicksort  Average Memory: 30kB  Peak Memory: 30kB
        Pre-sorted Groups: 4  Sort Method: quicksort  Average Memory: 34kB  Peak Memory: 36kB
        ->  Nested Loop  (cost=3.44..346.20 rows=117 width=125) (actual time=0.671..14.510 rows=790 loops=1)
              Join Filter: ((d2.state_name)::text = (s2.state_name)::text)
              Rows Removed by Join Filter: 39500
              ->  Index Scan using idx_region on state s2  (cost=0.14..3.90 rows=51 width=16) (actual time=0.023..0.050 rows=51 loops=1)
              ->  Materialize  (cost=3.30..253.08 rows=117 width=128) (actual time=0.003..0.201 rows=790 loops=51)
                    ->  Hash Join  (cost=3.30..252.50 rows=117 width=128) (actual time=0.138..7.564 rows=790 loops=1)
                          Hash Cond: (c2.district_id = d2.district_id)
                          ->  Nested Loop  (cost=0.57..249.44 rows=117 width=122) (actual time=0.062..7.154 rows=790 loops=1)
                                ->  Index Scan using idx_sub_prod on "CRMEvents" cr2  (cost=0.29..114.33 rows=117 width=236) (actual time=0.035..0.724 rov
                                      Index Cond: (("Sub-product")::text = 'Savings account'::text)
                                ->  Index Scan using client_pkey on client c2  (cost=0.28..1.15 rows=1 width=14) (actual time=0.007..0.007 rows=1 loops=790)
                                      Index Cond: ((client_id)::text = (cr2."Client_ID")::text)
                          ->  Hash  (cost=1.77..1.77 rows=77 width=14) (actual time=0.064..0.066 rows=77 loops=1)
                                Buckets: 1024  Batches: 1  Memory Usage: 12kB
                                ->  Seq Scan on district d2  (cost=0.00..1.77 rows=77 width=14) (actual time=0.013..0.031 rows=77 loops=1)
```

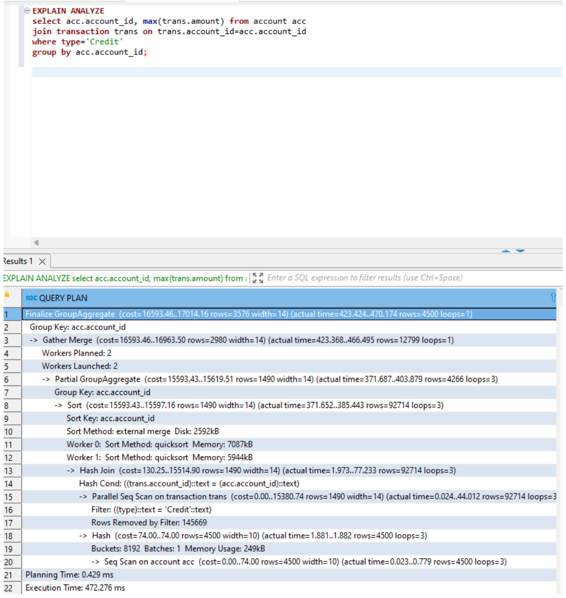


```
  ->  GroupAggregate  (cost=4.61..254.41 rows=4 width=15) (actual time=5.020..5.020 rows=1 loops=27)
        Group Key: s1.region
        ->  Hash Join  (cost=4.61..254.22 rows=30 width=7) (actual time=0.122..4.977 rows=201 loops=27)
              Hash Cond: (c1.district_id = d1.district_id)
              ->  Nested Loop  (cost=0.57..249.44 rows=117 width=4) (actual time=0.018..4.770 rows=790 loops=27)
                    ->  Index Scan using idx_sub_prod on "CRMEvents" cr1  (cost=0.29..114.33 rows=117 width=118) (actual time=0.011..0.524 rows=79
                          Index Cond: (("Sub-product")::text = 'Savings account'::text)
                    ->  Index Scan using client_pkey on client c1  (cost=0.28..1.15 rows=1 width=14) (actual time=0.005..0.005 rows=1 loops=21330)
                          Index Cond: ((client_id)::text = (cr1."Client_ID")::text)
              ->  Hash  (cost=3.79..3.79 rows=20 width=11) (actual time=0.071..0.071 rows=19 loops=27)
                    Buckets: 1024  Batches: 1  Memory Usage: 10kB
                    ->  Hash Join  (cost=1.80..3.79 rows=20 width=11) (actual time=0.034..0.055 rows=19 loops=27)
                          Hash Cond: ((d1.state_name)::text = (s1.state_name)::text)
                          ->  Seq Scan on district d1  (cost=0.00..1.77 rows=77 width=14) (actual time=0.003..0.009 rows=77 loops=27)
                          ->  Hash  (cost=1.64..1.64 rows=13 width=16) (actual time=0.022..0.022 rows=13 loops=27)
                                Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                ->  Seq Scan on state s1  (cost=0.00..1.64 rows=13 width=16) (actual time=0.007..0.013 rows=13 loops=27)
                                      Filter: ((region)::text = (s2.region)::text)
                                      Rows Removed by Filter: 38
Planning Time: 1.761 ms
Execution Time: 151.332 ms
```

Fig 28: After Indexing

cost=90.33 and Execution Time: 151.332 ms We can see there is significant decrease in the cost and execution time after Indexing.

2) **Find the maximum transaction of each account using Credit as transaction type:**


Results 1
EXPLAIN ANALYZE select acc.account_id, max(trans.amount) from a
Enter a SQL expression to filter results (use Ctrl+Space)

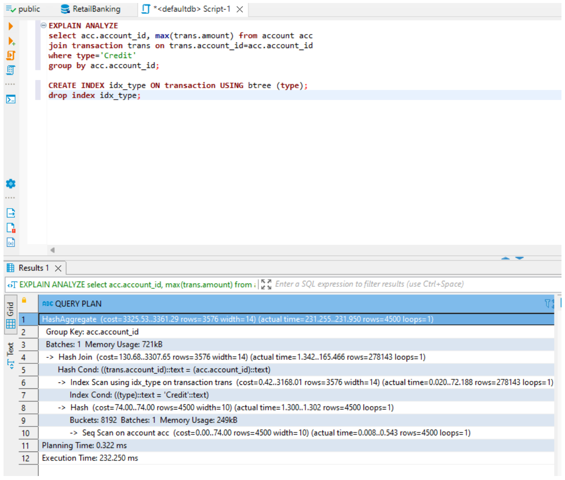
```
EXPLAIN ANALYZE
select acc.account_id, max(trans.amount) from account acc
join transaction trans on trans.account_id=acc.account_id
where type='Credit'
group by acc.account_id;
```

```
QUERY PLAN
Finalize GroupAggregate  (cost=16593.46..17014.16 rows=3576 width=14) (actual time=423.424..470.174 rows=4500 loops=1)
  Group Key: acc.account_id
  ->  Gather Merge  (cost=16593.46..16963.50 rows=2980 width=14) (actual time=423.368..466.495 rows=12799 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Partial GroupAggregate  (cost=15593.43..15619.51 rows=1490 width=14) (actual time=371.687..403.879 rows=4266 loops=3)
              Group Key: acc.account_id
              ->  Sort  (cost=15593.43..15597.16 rows=1490 width=14) (actual time=371.652..385.443 rows=92714 loops=3)
                    Sort Key: acc.account_id
                    Sort Method: external merge  Disk: 2592kB
                    Worker 0:  Sort Method: quicksort  Memory: 7087kB
                    Worker 1:  Sort Method: quicksort  Memory: 5944kB
                    ->  Hash Join  (cost=130.25..15514.90 rows=1490 width=14) (actual time=1.973..77.233 rows=92714 loops=3)
                          Hash Cond: ((trans.account_id)::text = (acc.account_id)::text)
                          ->  Parallel Seq Scan on transaction trans  (cost=0.00..15380.74 rows=1490 width=14) (actual time=0.024..44.012 rows=92714 loops=3
                                Filter: ((type)::text = 'Credit'::text)
                                Rows Removed by Filter: 145669
                          ->  Hash  (cost=74.00..74.00 rows=4500 width=10) (actual time=1.881..1.882 rows=4500 loops=3)
                                Buckets: 8192  Batches: 1  Memory Usage: 249kB
                                ->  Seq Scan on account acc  (cost=0.00..74.00 rows=4500 width=10) (actual time=0.023..0.779 rows=4500 loops=3)
Planning Time: 0.429 ms
Execution Time: 472.276 ms
```

Fig 28: Before Indexing

cost=16593.46 and Execution Time: 472.276 ms

Here as the transaction type is taking sequential scan and is causing the execution time of the query longer, we tried to index on transaction type column and see the results


public
RetailBanking
*<defaultdb> Script-1
Results 1
EXPLAIN ANALYZE select acc.account_id, max(trans.amount) from a
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text

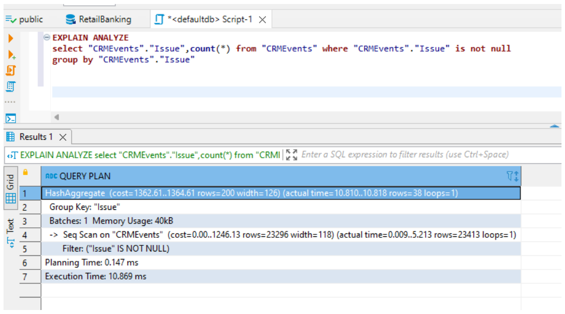
```
EXPLAIN ANALYZE
select acc.account_id, max(trans.amount) from account acc
join transaction trans on trans.account_id=acc.account_id
where type='Credit'
group by acc.account_id;

CREATE INDEX idx_type ON transaction USING btree (type);
drop index idx_type;
```

```
QUERY PLAN
HashAggregate  (cost=3325.53..3361.29 rows=3576 width=14) (actual time=231.255..231.950 rows=4500 loops=1)
  Group Key: acc.account_id
  Batches: 1  Memory Usage: 721kB
  ->  Hash Join  (cost=130.68..3307.65 rows=3576 width=14) (actual time=1.342..165.466 rows=278143 loops=1)
        Hash Cond: ((trans.account_id)::text = (acc.account_id)::text)
        ->  Index Scan using idx_type on transaction trans  (cost=0.42..3168.01 rows=3576 width=14) (actual time=0.020..72.188 rows=278143 loops=1)
              Index Cond: ((type)::text = 'Credit'::text)
        ->  Hash  (cost=74.00..74.00 rows=4500 width=10) (actual time=1.300..1.302 rows=4500 loops=1)
              Buckets: 8192  Batches: 1  Memory Usage: 249kB
              ->  Seq Scan on account acc  (cost=0.00..74.00 rows=4500 width=10) (actual time=0.008..0.543 rows=4500 loops=1)
Planning Time: 0.322 ms
Execution Time: 232.250 ms
```

Fig 28: After Indexing

cost=3325.53 and Execution Time: 232.250 ms We can see there is significant decrease in the cost and execution time after Indexing.

3) **Find the count of number of events registered for each Issue in crmevents which are not null**


public
RetailBanking
*<defaultdb> Script-1
Results 1
EXPLAIN ANALYZE select "CRMEvents"."Issue",count(*) from "CRMI
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text

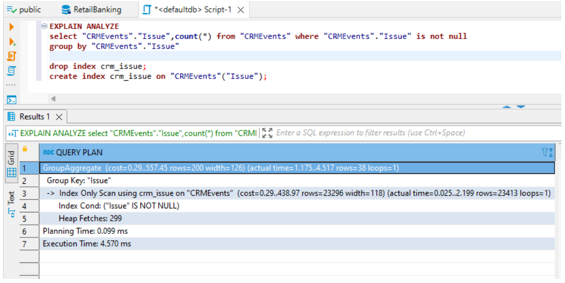
```
EXPLAIN ANALYZE
select "CRMEvents"."Issue",count(*) from "CRMEvents" where "CRMEvents"."Issue" is not null
group by "CRMEvents"."Issue"
```

```
QUERY PLAN
HashAggregate  (cost=1362.61..1364.61 rows=200 width=126) (actual time=10.810..10.818 rows=38 loops=1)
  Group Key: "Issue"
  Batches: 1  Memory Usage: 40kB
  ->  Seq Scan on "CRMEvents"  (cost=0.00..1246.13 rows=23296 width=118) (actual time=0.009..5.213 rows=23413 loops=1)
        Filter: ("Issue" IS NOT NULL)
Planning Time: 0.147 ms
Execution Time: 10.869 ms
```

Fig 28: Before Indexing

cost=1362.61 and Execution Time: 10.869 ms

Here as the crmevents issue is taking sequential scan and is causing the execution time of the query longer, we tried to index on crmevents ”issue” column and see the results



```
EXPLAIN ANALYZE
select "CRMEvents"."Issue",count(*) from "CRMEvents" where "CRMEvents"."Issue" is not null
group by "CRMEvents"."Issue"

drop index crm_issue;
create index crm_issue on "CRMEvents"("Issue");
```

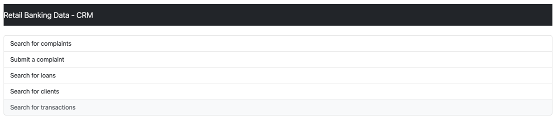
```
EXPLAIN ANALYZE select "CRMEvents"."Issue",count(*) from "CRMI
  QUERY PLAN
1 GroupAggregate  (cost=0.29..557.45 rows=200 width=126) (actual time=1.175..4.517 rows=38 loops=1)
2   Group Key: "Issue"
3   ->  Index Only Scan using crm_issue on "CRMEvents"  (cost=0.29..438.97 rows=23296 width=118) (actual time=0.025..2.199 rows=23413 loops=1)
4         Index Cond: ("Issue" IS NOT NULL)
5         Heap Fetches: 299
6 Planning Time: 0.099 ms
7 Execution Time: 4.570 ms
```

Fig 28: After Indexing

cost=0.29 and Execution Time: 4.570 ms
We can see there is significant decrease in the cost and execution time after Indexing.

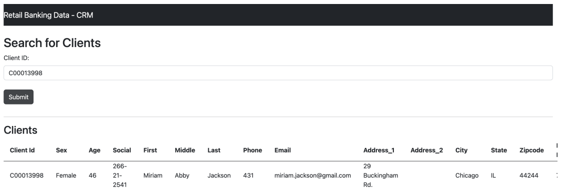
XI. Dashboard

The web application made as a part of the bonus task is up and available at https://lionfish-app-y3ij5.ondigitalocean.app/


Retail Banking Data - CRM

- Search for complaints
- Submit a complaint
- Search for loans
- Search for clients
- Search for transactions

Fig 29: Home Page


Retail Banking Data - CRM

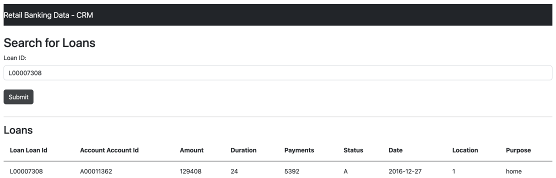
Search for Clients

Client ID: C00013998

Clients

Client Id	Sex	Age	Social	First	Middle	Last	Phone	Email	Address_1	Address_2	City	State	Zipcode
C00013998	Female	46	266-21-2541	Miriam	Abby	Jackson	431	miriam.jackson@gmail.com	29 Buckingham Rd.		Chicago	IL	44244

Fig 30: Search Clients


Retail Banking Data - CRM

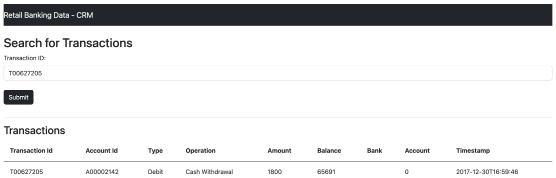
Search for Loans

Loan ID: L00007308

Loans

Loan Loan Id	Account Account Id	Amount	Duration	Payments	Status	Date	Location	Purpose
L00007308	A00011362	129408	24	5392	A	2016-12-27	1	home

Fig 31: Search Loans


Retail Banking Data - CRM

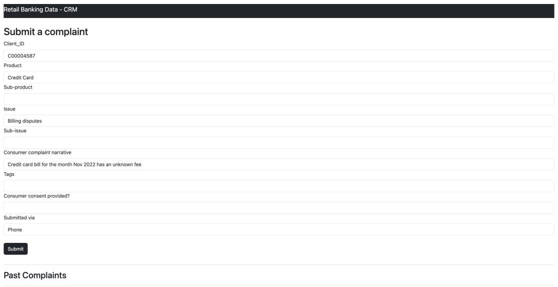
Search for Transactions

Transaction ID: T00627205

Transactions

Transaction Id	Account Id	Type	Operation	Amount	Balance	Bank	Account	Timestamp
T00627205	A00002142	Debit	Cash Withdrawal	1800	65691		0	2017-12-30T16:59:46

Fig 32: Search Transactions

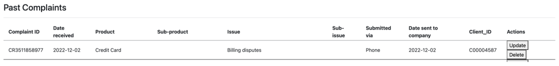

Retail Banking Data - CRM

Submit a complaint

Client_ID: C00004587
Product: Credit Card
Sub-product:
Issue: Billing disputes
Sub-issue:
Consumer complaint narrative: Credit card bill for the month Nov 2022 has an unknown fee
Tags:
Consumer consent provided?:
Submitted via: Phone

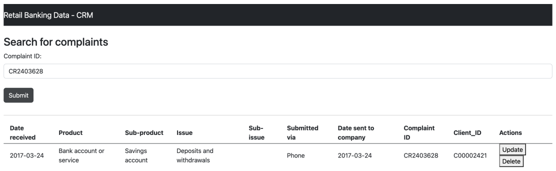
Past Complaints

Fig 33: Complaint submission Portal


Past Complaints

Complaint ID	Date received	Product	Sub-product	Issue	Sub-issue	Submitted via	Date sent to company	Client_ID	Actions
CR3511858977	2022-12-02	Credit Card		Billing disputes		Phone	2022-12-02	C00004587	Update Delete

Fig 34: Complaint submission Portal — Submitted Complaint


Retail Banking Data - CRM

Search for complaints

Complaint ID: CR2403628

Date received	Product	Sub-product	Issue	Sub-issue	Submitted via	Date sent to company	Complaint ID	Client_ID	Actions
2017-03-24	Bank account or service	Savings account	Deposits and withdrawals		Phone	2017-03-24	CR2403628	C00002421	Update Delete

Fig 35: Search Complaints

Future Scope

∗ **Website and Public Portal** - We have already built a CRM dashboard that would help the bank officials manage behind-the-scenes. A future prospect is to add a public-facing customer portal to help audience at a broader scale interact with Eagle National Bank. Our database and design would ensure that there is no redundant data across the tables and maintain database's integrity.

∗ **Login Credentials** - Personalized dashboards can be secured with login credentials and distributed to officials to prevent fraudulent activities.

∗ **Trigger** - We can add Triggers at suitable places to perform routine cleanup activities to maintain the sanity of the database.

References

[1] "Retail Banking Demo Data - Dataset by Lpetrocelli." data.world, May 3, 2022. https://data.world/lpetrocelli/retail-banking-demo-data.