USE CASE STUDY REPORT

Group No.: Group 7

Student Names: Sahita Bonthu and Divya Babulal Shah

Executive Summary:

This study is focused on the development and implementation of a comprehensive database schema tailored for a Caller Scan system. The overarching goal is to provide a comprehensive platform for managing user interactions, focusing on calls, profile views, and subscription management. The schema incorporates multiple tables, including user details, login history, subscription types, call logs, and spam reports, with features such as profile views, notifications, and an automated spam blocking system. The data modeling aspect involves intricate design components. User Management utilizes a User table linked to email addresses, while the Subscription table tracks details with a trigger for dynamic end dates. Profile views are recorded in the ProfileView table, generating notifications through the Profile_View_Notification table. The Location and Contacts tables store user-related information, and the Spam table logs reported spam, managed by the BlockList table. The CallLog table records user calls, with a trigger for automatic call blocking based on spam reports.

The EER and UML diagrams were modelled, followed by the mapping of the conceptual model to a relational model with the required primary and foreign keys. This database was then implemented fully MySQL and a prototype with eleven tables was implemented on Neo4j NoSQL graph database to study the feasibility of this database in a NoSQL environment. The created database was a great success, and by connecting it to Python, the analytics capabilities are immense, some of which have been shown in the study. Recommendations for system optimization and security include regular monitoring, additional security measures, and seamless user interface integration. Ongoing maintenance and enhancements are advocated to ensure continual effectiveness and security.

In conclusion, the Caller Scan database schema reflects a meticulous approach to managing user interactions, subscriptions, and automation, with ongoing maintenance essential for sustained effectiveness and security.

I. Introduction

In the contemporary digital landscape, characterized by the ubiquitous use of smartphones and the surge in user-generated content, the challenges associated with data security, privacy concerns, and the proliferation of cyber threats have become increasingly pronounced. Recognizing the significance of these issues, this project aims to delve into the development of a comprehensive Database Management System (DBMS) tailored to address both traditional and modern challenges in user data management. The opportunity lies in crafting a solution that not only streamlines conventional user data management processes but also actively confronts contemporary issues such as fraud detection, spam prevention, and enhanced user privacy.

The problem at hand revolves around the widespread cyber threats prevalent in the era of smartphone usage, coupled with an elevated risk of data breaches and privacy concerns. Users find themselves navigating through the complexities of potential fraudsters, spam calls, and the imperative need for heightened privacy in their online interactions. The overarching goal of this study is to design and implement a DBMS that not only tackles conventional challenges in user data management but also addresses contemporary issues. The envisioned solution encompasses various requirements, including efficient user management, streamlined subscription processes, activity tracking, and the implementation of advanced features such as Caller Identification, Spam Detection, Profile Privacy, and a User-driven Rating System.

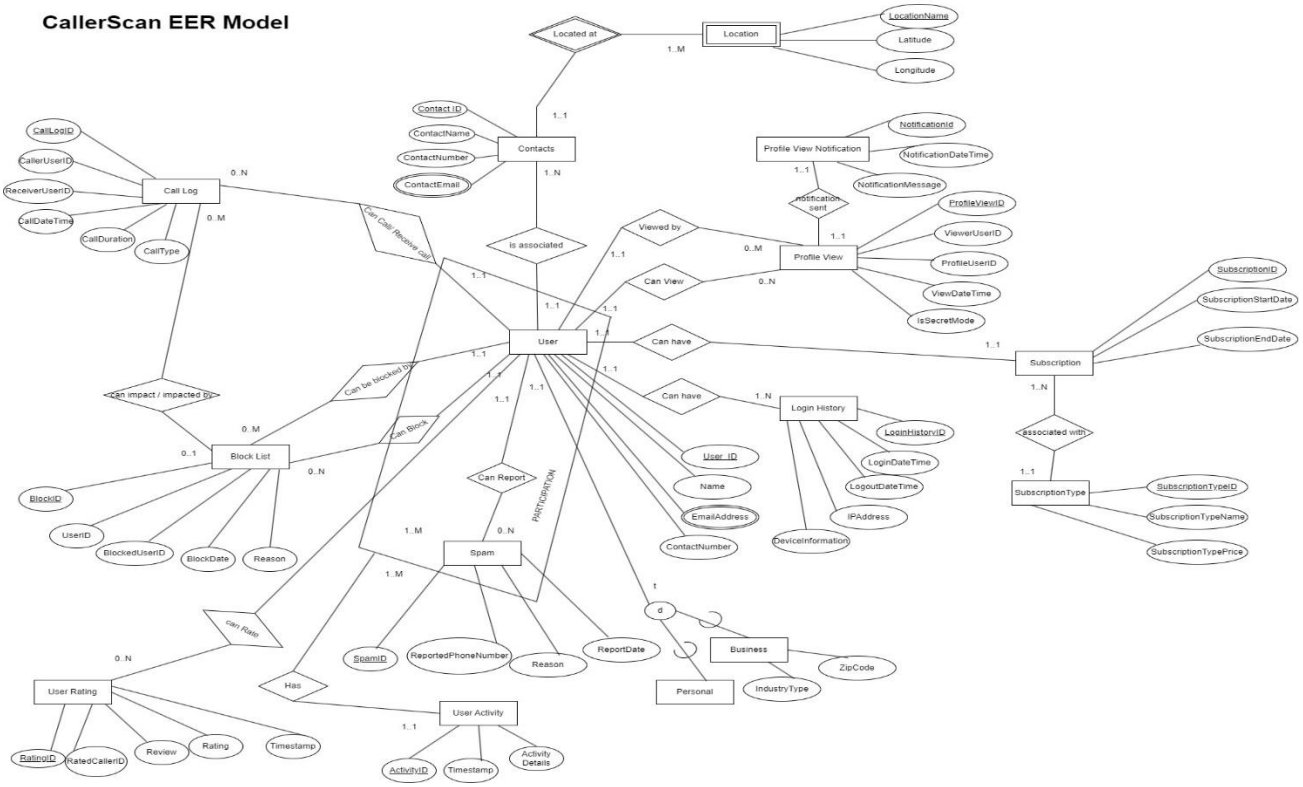
Moreover, the project acknowledges additional requirements crucial to the system's functionality:

- One user can have multiple contacts; each contact is associated with one user.
- One user can make multiple calls; each call log entry has one caller user.
- One user can block multiple users; each block list entry has one blocking user.
- One user can report multiple phone numbers as spam; each spam report is associated with multiple users.
- One contact can have multiple location records; each location record is associated with one contact.

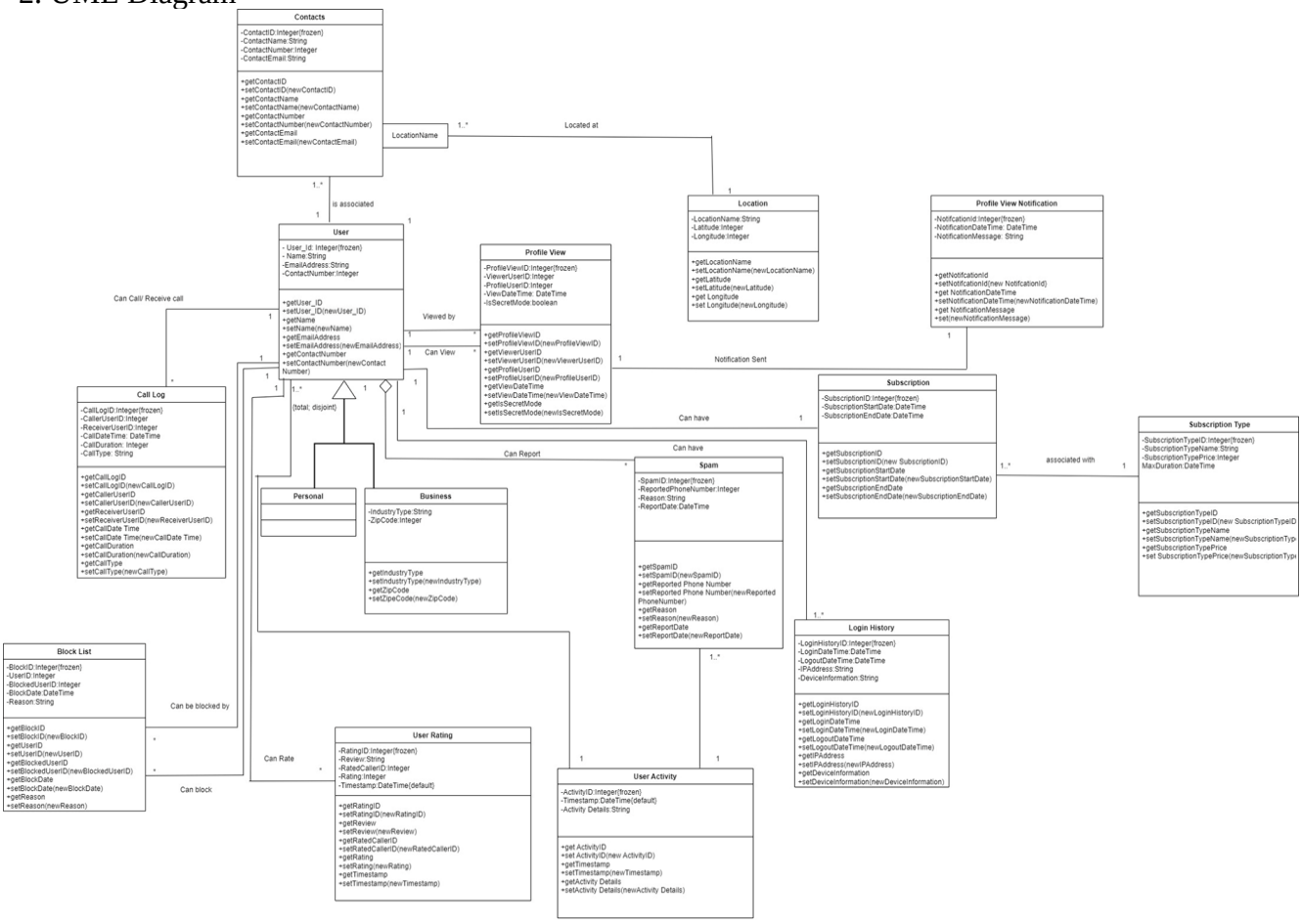
By addressing these comprehensive requirements, the DBMS endeavors to offer a holistic solution, ensuring not only efficient user data management but also seamless interaction across multiple facets of user communication and security.

II. Conceptual Data Modeling

1. EER Diagram



2. UML Diagram



III. Mapping Conceptual Model to Relational Model

Primary keys are underlined and foreign keys are in italics

Primary keys are NOT NULL by default.

- User(User_ID, Name , ContactNumber)
- User-Email(EmailAddress, *User_ID*)
  - User_ID: foreign key refers to User_ID in relation User : NULL not allowed
- Login History(LoginHistoryID, LoginDateTime, LogoutDateTime, IPAddress, DeviceInformation, *Login_User_ID*)
  - Login_User_ID: foreign key refers to User_ID in relation User : NULL not allowed
- Subscription(SubscriptionID, SubscriptionStartDate, SubscriptionEndDate, *Subscription_User_ID, SubscriptionTypeID*)
  - Subscription_User_ID : foreign key refers to User_ID in relation User : NULL not allowed
  - SubscriptionTypeID : foreign key refers to SubscriptionTypeID in relation User : NULL not allowed
- SubscriptionType(SubscriptionTypeID,SubscriptionTypeName, SubscriptionTypePrice)

- ProfileView(ProfileViewID, *ViewerUserID*, *ProfileUserID*, ViewDateTime, IsSecretMode)
  - ViewerUserID: foreign key refers to User_ID in relation User : NULL not allowed
  - ProfileUserID: foreign key refers to User_ID in relation User : NULL not allowed
- Profile View Notification(NotificationId, NotificationDateTime, NotificationMessage, *ProfileViewID*)
  - ProfileViewID: foreign key refers to ProfileViewID in relation User : NULL not allowed
- Location(LocationName, Latitude, Longitude, *Contact_ID*)
  - Contact_ID: foreign key refers to User_ID in relation User : NULL not allowed
- Personal(*Personal_User_ID)*
  - Personal_User_ID: foreign key refers to User_ID in relation User : NULL not allowed
- Business(*Business_User_ID*, IndustryType, ZipCode)
  - Business_User_ID: foreign key refers to User_ID in relation User : NULL not allowed
- CallLog(CallLogID, *CallerUserID*, *ReceiverUserID*, CallDateTime, CallDuration, CallType, *Call_Log_BlockID*)
  - CallerUserID : foreign key refers to User_ID in relation User: NULL not allowed
  - ReceiverUserID : foreign key refers to User_ID in relation User: NULL not allowed
  - Call_Log_BlockID: foreign key refers to BlockID in relation User: NULL allowed
- BlockList(BlockID, *UserID*, *BlockedUserID*, BlockDate, Reason)
  - UserID : foreign key refers to User_ID in relation User: NULL not allowed
  - BlockedUserID : foreign key refers to User_ID in relation User: NULL not allowed
- UserRating(RatingID, Review, Rating, Timestamp, *RatedCallerID*, *User_ID*)
  - User_ID: foreign key refers to User_ID in relation User: NULL not allowed
  - RatedCallerID : foreign key refers to User_ID in relation User: NULL not allowed
- Spam(SpamID, ReportedPhoneNumber, Reason, ReportDate)
- Contacts(Contact_ID, ContactName, ContactNumber, *ContactsUser_ID*)
  - ContactsUser_ID : foreign key refers to User_ID in relation User: NULL not allowed
- Contact-Email(ContactEmail, *Contact_ID*)
  - Contact_ID: foreign key refers to User_ID in relation User : NULL not allowed
- Paticipation(*P_SpamID*, *P_User_ID*, *P_ActivityID*)
  - P_SpamID: foreign key refers to SpamID in relation Spam: NULL not allowed
  - P_User_ID: foreign key refers to User_ID in relation User: NULL not allowed
  - P_Activity_ID: foreign key refers to Activity_ID in relation User: NULL not allowed
- UserActivity(ActivityID, Timestamp, Type)

IV. Implementation of Relational Model via MySQL and NoSQL

MySQL Implementation:

The database was created in MySQL and the following queries were performed:

Query 1: Find the total count of logins for each user, but only show users who have logged in more than 10 times.

```
SELECT Login_User_ID, COUNT(*) AS LoginCount
FROM login_history
GROUP BY Login_User_ID
HAVING LoginCount > 10;
```

Login_User_ID	LoginCount
17	15

Query 2: Retrieve the names of users who have subscribed to the "Monthly" subscription type and order the results by their names.

```
SELECT User.Name FROM User
JOIN Subscription
ON User.User_ID = Subscription.Subscription_User_ID
JOIN SubscriptionType
ON Subscription.SubscriptionTypeID =
SubscriptionType.SubscriptionTypeID
WHERE SubscriptionType.SubscriptionTypeName =
'Monthly' ORDER BY User.Name;
```

Name
Abbie Friesen
Avery Hahn
Beatrice Muller
Brandi King
Candido Sipes

Query 3: Retrieve call logs for the user named 'Marshall Farrell V' based on their user ID in the CallLog table.

```
SELECT CallLogID,CallerUserID,
ReceiverUserID,CallDateTime, CallDuration,
CallType FROM CallLog
WHERE CallerUserID IN (SELECT User_ID
FROM User WHERE Name = 'Marshall Farrell
V');
```

CallLogID	CallerUserID	ReceiverUserID	CallDateTime	CallDuration	CallType
1	31	1	2001-09-12 15:37:28	1743	incoming call
31	31	1	2023-11-11 11:38:23	490	outgoing call
NULL	NULL	NULL	NULL	NULL	NULL

Query 4: Find the users who have blocked other users and received ratings.

```
SELECT User.User_ID, User.Name
FROM User
WHERE User.User_ID IN (
    SELECT DISTINCT BlockList.UserID
    FROM BlockList
    WHERE BlockList.UserID = User.User_ID
) AND User.User_ID IN (
    SELECT DISTINCT UserRating.RatedCallerID
    FROM UserRating);
```

User_ID	Name
1	Kailey Bogan
2	Dr. Stanford Huels V
3	Brenna Monahan DDS
4	Rick Connelly Jr.
5	Candido Sipes

Query 5: Find all users who have made or received more than 10 calls in the month of November 2023, grouped by subscription type.

```
SELECT st.SubscriptionTypeName, COUNT(u.User_ID) AS
UserCount
FROM User u
INNER JOIN Subscription s ON u.User_ID =
s.Subscription_User_ID
INNER JOIN SubscriptionType st ON s.SubscriptionTypeID
= st.SubscriptionTypeID
WHERE EXISTS (
   SELECT 1
   FROM CallLog c
   WHERE (c.CallerUserID = u.User_ID OR
c.ReceiverUserID = u.User_ID)
   AND YEAR(c.CallDateTime) = 2023
   AND MONTH(c.CallDateTime) = 11 )
GROUP BY st.SubscriptionTypeName
HAVING UserCount > 10;
```

SubscriptionTypeName	UserCount
Free	22
Monthly	33
Annual	45

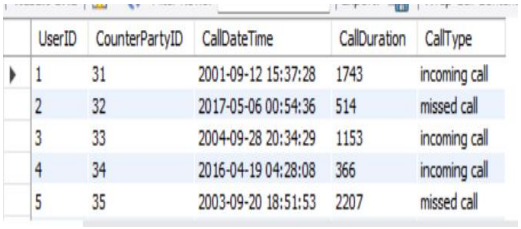
Query 6: Retrieve CallerUserID from the CallLog table where the call duration for at least one call is greater than 10 minutes and where all call durations for that caller are greater than 1 minute.

```
SELECT DISTINCT c1.CallerUserID, u.Name
FROM CallLog c1
JOIN User u ON c1.CallerUserID = u.User_ID
WHERE c1.CallDuration > 10 * 60 AND c1.CallerUserID = ANY (
   SELECT c2.CallerUserID
   FROM CallLog c2
   WHERE c1.CallerUserID = c2.CallerUserID
   )
AND 1 * 60 < ALL (
   SELECT c3.CallDuration
   FROM CallLog c3
   WHERE c1.CallerUserID = c3.CallerUserID
   );
```

CallerUserID	Name
31	Marshall Farrell V
33	Kaitlyn Dooley
35	Allie Blick
36	Lottie Collins
37	Ryleigh Grant

Query 7: Retrieve information on calls made by and received by users who are in the block list, combining and presenting relevant details from the CallLog and BlockList tables.

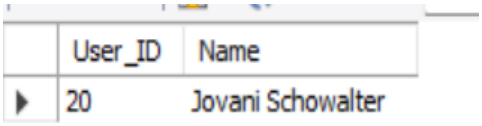
```
SELECT CallerUserID AS UserID,
   ReceiverUserID AS CounterPartyID,
   CallDateTime,
   CallDuration,
   CallType
FROM
   CallLog
WHERE CallerUserID IN (SELECT UserID FROM
BlockList)
UNION
-- Calls received by blocked users
SELECT ReceiverUserID AS UserID,
   CallerUserID AS CounterPartyID,
   CallDateTime,
   CallDuration,
   CallType
FROM
   CallLog
WHERE
   ReceiverUserID IN (SELECT UserID FROM
BlockList);
```



UserID	CounterPartyID	CallDateTime	CallDuration	CallType
1	31	2001-09-12 15:37:28	1743	incoming call
2	32	2017-05-06 00:54:36	514	missed call
3	33	2004-09-28 20:34:29	1153	incoming call
4	34	2016-04-19 04:28:08	366	incoming call
5	35	2003-09-20 18:51:53	2207	missed call

Query 8: Find the users who have viewed profiles in secret mode more than 5 times.

```
SELECT User.User_ID, User.Name
FROM User
JOIN (SELECT ViewerUserID, COUNT(*) AS
SecretViewCount
     FROM ProfileView
     WHERE IsSecretMode = 1
     GROUP BY ViewerUserID
     HAVING SecretViewCount > 5) AS SubQuery
ON User.User_ID = SubQuery.ViewerUserID;
```



User_ID	Name
20	Jovani Schowalter

NoSQL Implementation:

Eleven tables (users, userRatings, subscriptionTypes, subscriptions, spams, profileViewNotifications, profileViews, businesses, callLogs, contacts, locations) have been created in Neo4j playground.

NoSQL's MongoDB queries are as follows:

Query 1. Retrieve a User by Contact Number:

```
db.users.findOne({ ContactNumber: "+4445556666" });
```

Result


```
{
        "_id" : 4,
        "Name" : "David Miller",
        "ContactNumber" : "+4445556666",
        "User_Emails" : [
                {
                        "EmailAddress" : "david.miller@example.com"
                },
                {
                        "EmailAddress" : "david@gmail.com"
                }
        ]
}
```


Query 2. Retrieve Call Logs for a Specific User with Detailed Information:

```
db.callLogs.aggregate([
{$match: {$or: [
{ CallerUserID: 14 },
{ ReceiverUserID: 14 }],},},
{$lookup: { from: "users", localField:"CallerUserID",
foreignField: "_id",as: "callerUser",},},
{$lookup: { from: "users",localField:"ReceiverUserID",
foreignField: "_id", as: "receiverUser", },},
{ $project:
{ CallDateTime: 1,
CallDuration: 1,
CallType: 1,
"callerUser.Name": 1,
"receiverUser.Name": 1,
},},]);
```

Result


```
"_id" : 13, "CallDateTime" : ISODate("2023-09-15T11:45:00Z"), "CallDuration" : 240, "CallType"
"_id" : 18, "CallDateTime" : ISODate("2023-12-01T18:30:00Z"), "CallDuration" : 180, "CallType"
```

Query 3. Calculate Average Rating Given by Users:

```
var mapFunction = function () {
  emit(this.RatedCallerID, this.Rating);
};
var reduceFunction = function (key, values) {
  return Array.avg(values);
};
db.userRatings.mapReduce(
  mapFunction,
  reduceFunction,
  { out: "averageRatings" }
).find();
```

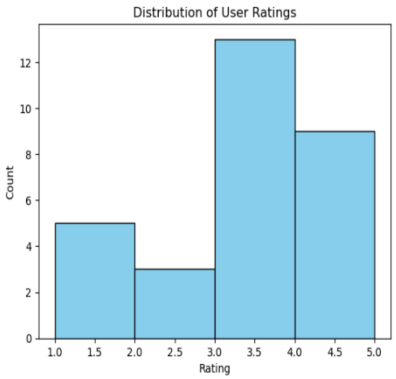
Result

```
{ "_id" : 2, "value" : 4 }
{ "_id" : 3, "value" : 4 }
{ "_id" : 4, "value" : 5 }
{ "_id" : 5, "value" : 4 }
{ "_id" : 6, "value" : 5 }
{ "_id" : 7, "value" : 4 }
{ "_id" : 8, "value" : 4 }
{ "_id" : 9, "value" : 5 }
{ "_id" : 10, "value" : 4 }
{ "_id" : 11, "value" : 4 }
{ "_id" : 12, "value" : 5 }
{ "_id" : 13, "value" : 4 }
{ "_id" : 14, "value" : 5 }
{ "_id" : 15, "value" : 4 }
{ "_id" : 16, "value" : 4.5 }
{ "_id" : 17, "value" : 4.5 }
{ "_id" : 18, "value" : 4.5 }
{ "_id" : 19, "value" : 4.5 }
{ "_id" : 20, "value" : 4.5 }
```

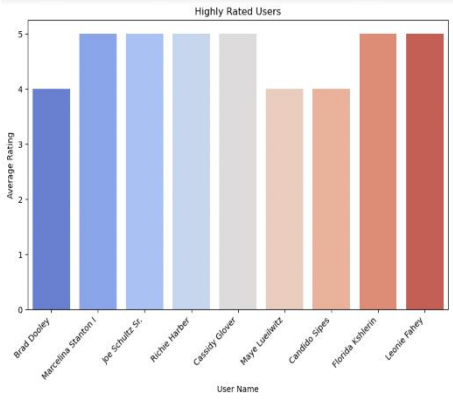
V. Database Access via R or Python

The database is accessed using Python and the visualization of analyzed data is shown below. The connection of MySQL to Python is done using pymysql.connect(). We also used matplotlib.pyplot and seaborn library to visualize data.

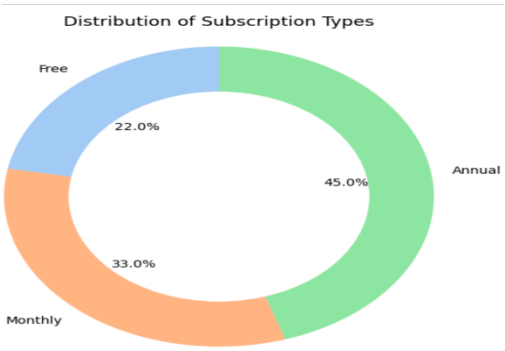
Graph 1: Histogram for user ratings



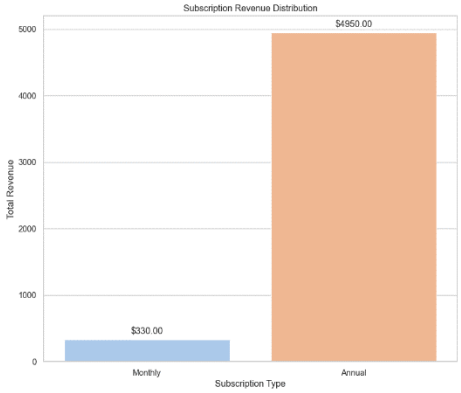
Graph 2: Bar Plot for Highly Rated Users



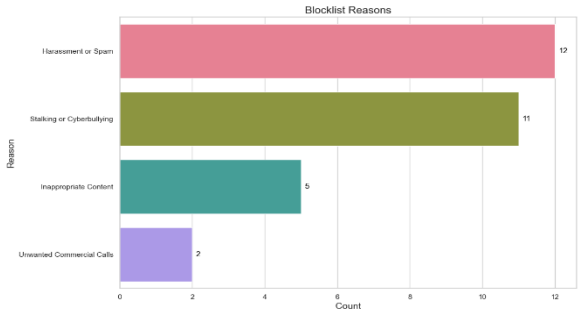
Graph 3: Distribution of subscription types



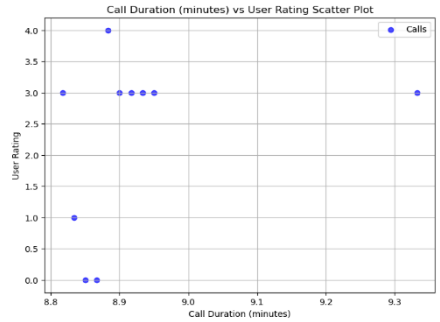
Graph 4: Subscription Revenue Distribution



Graph 5: Bar Chart for Blocklist Reasons



Graph 6: Call duration vs User Ratings.



VI. Summary and recommendation

The Caller Scan system's database schema, crafted for effective user interaction management, subscription handling, and communication management, presents various strengths, including comprehensive functionality, adaptability in MySQL and Neo4j NoSQL environments, and robust analytics facilitated by Python integration. However, the system needs to address security concerns through additional measures, emphasizing regular monitoring to safeguard user data. Seamless user interface integration requires attention to enhance overall user experience. A crucial recommendation involves the implementation of data governance measures to elevate data quality and security within the database. By fortifying data governance, the system can ensure the integrity of information and enhance overall data security.