**Database Design and Implementation for Real Time Chat Application**

**Final Project Report**

**Subject: - ISM6218 ADVANCED DATABASE MANAGEMENT**

Dinesh Datta Pendyala ( U96239813 )

Goutham Reddy Mandadi ( U13039494 )

Venkata Sai Prasanna Kumar Tummala ( U92076820 )

[Ashwin Kumar Reddy Guduru](https://usflearn.instructure.com/courses/1732246/users/4923279) ( U74844599 )

[Sri Sai Teja Balusu](https://usflearn.instructure.com/courses/1732246/users/4953933) ( U25008106 )

Yaswanth Chandolu ( U03348669 )

**Major in**

**BUSINESS ANALYTICS AND INFORMATION SYSTEMS**

**Under the guidance of**

**Dr. Don Berndt**

A picture containing text

Description automatically generated

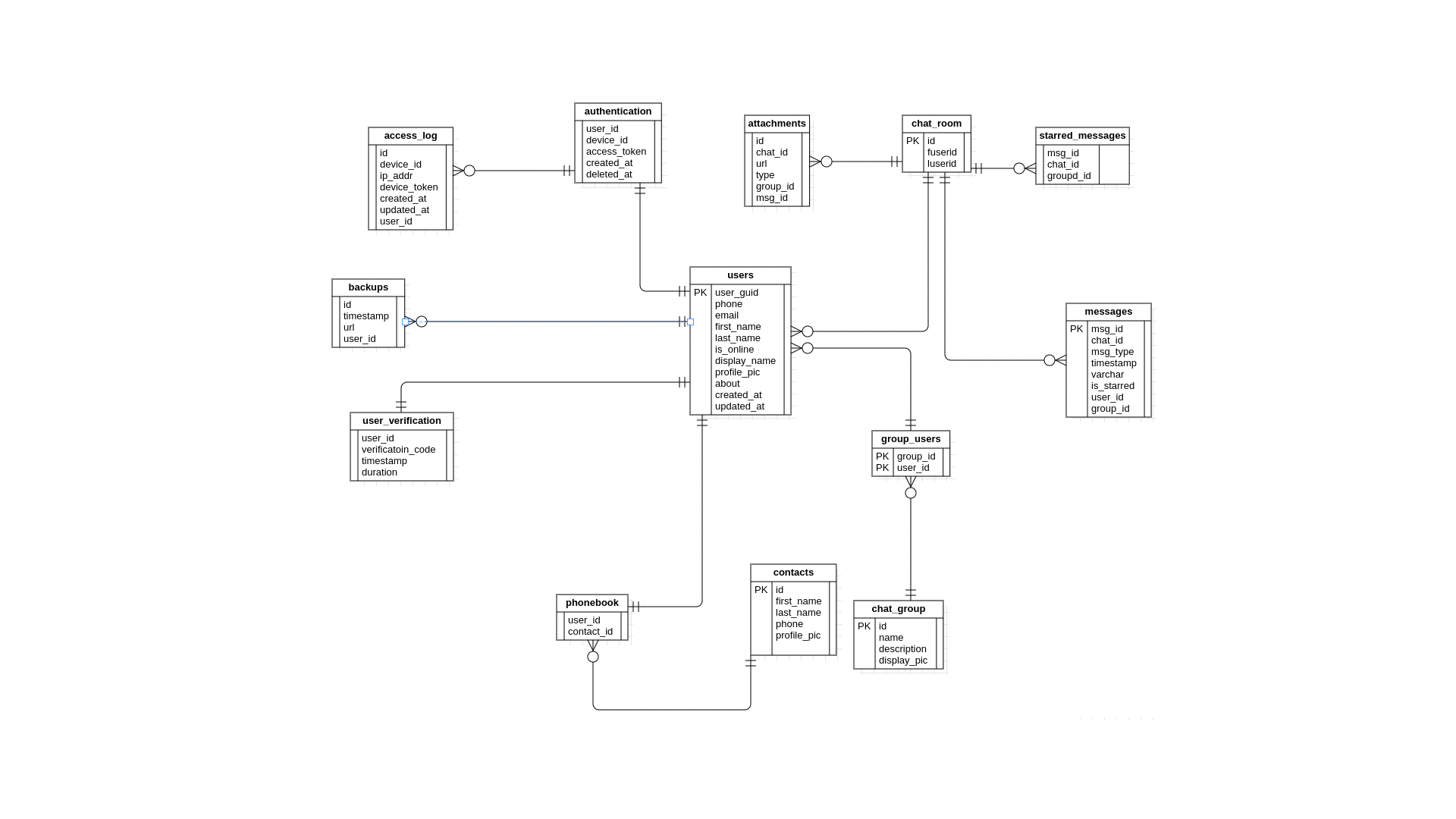
**MUMA COLLEGE OF BUSINESS**

**UNIVERSITY OF SOUTH FLORIDA**

**Introduction:**

The real time chat application has many complex interacting components. Each component needs to store different data types which may be used by various components. A careful design of database schema is critical to this kind of applications. The whole schema tables can be divided into three parts authentication, application and backup. The authentication part handles storage of current session,otp storage and access logs. The applications handles messages and groups interaction storage. The backup stores the location of backups. With indexing and normalization we achieved optimized query execution. This database can be deployed in production with a more additional tables.

**Part-1 Design Overview**



The Users table is the central part of the design with user\_guid as primary key. Each User in the table is linked with phonebook table, group\_users, chat\_room, authentication, user\_verification and backups as every table has user\_guid as dependent data in the table. The authentication is connected to access\_log in order to record the access log information when a user logs in. user\_verification table is used to store the generated one time code for a limited amount of time. Each chat\_room entry contains two users involved in conversation. Each message is associated with a particular chat\_room id and user\_id and group\_id. Starred\_messages has all the messages starred or marked favourite by a user in particular chat \_room or group. chat\_group facilitates any groups created by users and is connected to users using group\_users. Attachments table has the url of attachments and is in chat\_room table where it belongs. chat\_group table has description of the particular chat group. Backups table has an url where the backup of the user chat is stored.

**Part-2 Data base Tables**

1. Users:

USER\_GUID: The unique user id generated to every user after they sign up.[primary key]

PHONE: phone number of user.

EMAIL: email of user

FIRST\_NAME: first name of user

LAST\_NAME: last name of user

IS\_ONLINE: boolean to indicate whether user is online

DISPLAY\_NAME: display name of user

PROFILE\_PIC: profile picture url of user

ABOUT: A few lines about user.

CREATED\_AT: timestamp of when the row is created

UPDATED\_AT: time stamp of when the row is updated

2. Contacts:

ID: The unique contact\_id for each contact

FIRST\_NAME: First name of contact

LAST\_NAME: Last name of contact

PHONE: Phone number of contact

PROFILE\_PIC: URL of Profile picture

3. PhoneBook:

USER\_ID: Id of user

CONTACT\_ID: Id of contact

4. Group\_users:

Group\_ID: group id of the group.

User\_ID: user involved in the group

5. Chat\_Group:

ID: The unique id of the group

NAME: Name of the group

DESCRIPTION: Description of the group

DISPLAY\_PIC: Url of display picture of the group

6. Chat\_room:

room\_ID: Id of room

FUSERID: First user of chat room

LUSERID: Last user of chat room

7. Starred\_Message:

MSG\_ID: starred Message id

CHAT\_ID: chat id in which the message is starred

GROUP\_ID: group in which the message is starred

8. User\_Verification

USER\_ID: Id of user login in

VERIFICATION\_CODE: code generated for user login

TIMESTAMP: timestamp at which code is generated

DURATION: Duration of code validity

9. Authentication

USER\_ID:Id of user logged in

DEVICE\_ID: Device through which user tried to login

ACCESS\_TOKEN: Access token generated for the user session

CREATED\_AT: created at field

DELETED\_AT: deleted at field

10. Contacts

ID: Contact id

FIRST\_NAME: First name of contact

LAST\_NAME: Last name of contact

PHONE: phone number of contact

PROFILE\_PIC: Url of profile pic

11. Attachments

ID: Attachment id

TYPE: Type of Attachment

MSG\_ID: Message id of the attachment

CHAT\_ID: chat room id in which attachment is sent

GROUP\_ID: group id in which attachment is sent

URL: Url of the stored attachment.

12. Access\_Log

ID: Access log id

DEVICE\_ID: device id through which the user logged in

IP\_ADDR: Ip address of the user login

DEVICE\_TOKEN: Device token generated to identify the device

USER\_ID: User id of the user logging in

CREATED\_AT: timestamp of when the row is created

DELETED\_AT: time stamp of when the row is deleted

13. Messages

MSG\_ID: Message id

CHAT\_ID: chat room id in which the message is sent

MSG\_TYPE: Type of message

TIMESTAMP: Timestamp of message

MSG: Text description of message

IS\_STARRED: Boolean indicating

USER\_ID: User id of user who sent the messages

GROUP\_ID: Group id of user

**Part-3 Key constraints**

|  |  |  |
| --- | --- | --- |
| **Tables** | **Attributes** | **Constraint Name** |
| users-chat\_room | USER\_GUID<=>Userid | FK\_USER\_CHAT |
| users-group\_users | USER\_GUID<=>User\_id | FK\_USER\_GROUP |
| users-user\_verfication | User\_GUID<=>USER\_ID | FK\_USER\_VERIFICATION |
| users-backups | USER\_GUID<=>USER\_ID | FK\_USER\_BACKUP |
| users-authentication | USER\_GUID<=>USER\_ID | FK\_USER\_AUTH |
| chat\_group-group\_users | ID<=>GROUP\_ID | FK\_CGRP\_GRP |
| messages-chat\_room | ID<=>CHAT\_ID | FK\_MSG\_CTR |
| chat\_room-starred-messages | ID<=>CHAT\_ID | FK\_CTR\_SMSG |
| authentication-access\_log | USER\_ID<=>USER\_ID | FK\_AUTH\_ALOG |

**Part-4 Database Normalization**

Normalization process of eliminating data redundancy and enhances data integrity int the table. In chatting application, to avoid data redundancy we breakdown contact table into Phone book and contacts. In Phonebook, each user\_id will have multiple contact\_id. In contacts, contact\_id, first\_name, last\_name, phone and profile\_pic. This will avoid redundancy.

Initially in the Contacts dataset, we have 6 different columns with (ID, First Name, Last Name, Phone, Profile Pic and User ID). Each Multiple ID’s will have multiple user id’s. Followed by these user id’s will have various information to store which was leading to redundancy. Now, we have broke it down to two different tables with Contacts and Phone book.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **First Name** | **Last Name** | **Phone** | **Profile Pic** | **User ID** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **First Name** | **Last Name** | **Phone** | **Profile Pic** |

|  |  |
| --- | --- |
| **ID** | **User ID** |

**Part-5 Indexing**

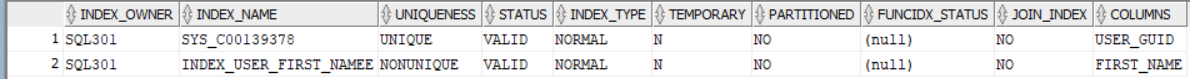
Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexing is optional structure used to improve query performance. For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers. In similar fashion, we have created index.

In the chatting application dataset, we have created below indexes to improve the performance of various tables.

Index 1: Have created indexing on USER\_GUID and FIRST\_NAME columns.

CREATE INDEX INDEX\_USER\_FIRST\_NAMEE ON USERSS("FIRST\_NAME")

CREATE INDEX INDEX\_USER\_USER\_GUID ON USERSS("USER\_GUID")



Below are the two scenarios where we have used indexing to check cost and runtime before and after indexing creation. After indexing creation, runtime and cost are marginally reduced.

Snapshot 1: Before indexing creation

Graphical user interface, text, application, email

Description automatically generated

Snapshot 2: After indexing creation

Graphical user interface, text, application

Description automatically generated

Index 2: Have created indexing on USER\_GUID and FIRST\_NAME columns.

**Part-6 Partitioning**

Partitioning is a effective method, which can improve query performance by dividing table into pieces that can be accessed separately. A table can be partitioned by one or more attributes which are widely used in queries. In chatting application, we have applied partitioning for year column in user table which is used very frequently to check active users. Below are the ways to create partitioning.

* Partition can be created by “CREATE TABLE statement with a PARTITION BY clause.”
* Resulting table will have series of active users over the years by decade.
* Any query using release year predicate, will access only relevant partitions.

**Partition creation in Oracle SQL developer for users table:**

PARTITION BY RANGE (CREATED\_AT) (

PARTITION p1970s VALUES LESS THAN (1980),

PARTITION p1980s VALUES LESS THAN (1990),

PARTITION p1990s VALUES LESS THAN (2000),

PARTITION p2000s VALUES LESS THAN (2010),

PARTITION p2010s VALUES LESS THAN (2020),

PARTITION p2020s VALUES LESS THAN (2030));

**Partition Output:**

Partition has been created on “CREATED\_AT” Column with six different partition names (refer snapshot below).

Text

Description automatically generated

**Sample query and Partition usage:**

From the below snapshot, we have used partition created earlier. An execution plan below showing range operation accessing 7th Partition.

Graphical user interface, application

Description automatically generated

**Part- 7 Query writing**

In Query writing, we have made use of different combinations of tables and written queries.

1. Top 20% users who got verification codes more than 3 times

**Query:**

SELECT DISTINCT USER\_GUID,first\_name,last\_name,COUNT(VERIFICATION\_CODE) AS Activity

FROM USERS U

INNER JOIN USER\_VERIFICATION UU

ON UU.USER\_ID=U.USER\_GUID

GROUP BY USER\_GUID,first\_name,last\_name

ORDER BY ACTIVITY DESC

FETCH FIRST 20 PERCENT ROWS ONLY;

**Output:**

Graphical user interface, text

Description automatically generated

2. Top 10% users who logged in more than 3 devices

**Query:**

SELECT USER\_GUID,FIRST\_NAME,COUNT(DEVICE\_ID) as devices

from users UU

inner join AUTHENTICATION AA

ON AA.USER\_ID=UU.USER\_GUID

GROUP BY USER\_GUID,FIRST\_NAME

ORDER BY DEVICES DESC

FETCH FIRST 10 PERCENT ROWS ONLY;

**Output:**

Text

Description automatically generated

3. Find the users who were active in Last 10 years.

**Query:**

SELECT DISTINCT USER\_GUID,DISPLAY\_NAME

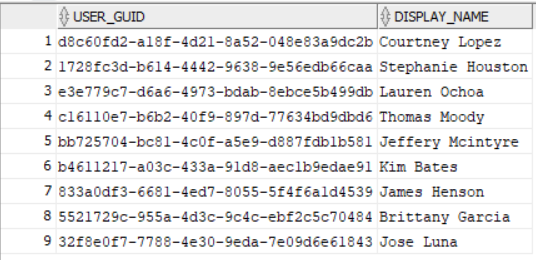
FROM USERS UU

INNER JOIN AUTHENTICATION AA

ON AA.USER\_ID = UU.USER\_GUID

WHERE CREATED\_AT BETWEEN CURRENT\_DATE -1000 AND CURRENT\_DATE

**Output:**



**Part- 8 Stored Procedures**

Stored procedure is a re-usable SQL code that we can save, so the code can be reused over and over again. If we have, SQL code that need to be re-used again and again. We can save that code and use it whenever it is required. It is group of SQL statements which accepts some input in the form of parameters and performs task and returns a value.

Below are the stored procedures which we have created for Chatting application database.

**Procedure 1:**

**-- no of messages in each message type sent by each user**

**CRATE PROCEDURE SQL301.MSGTYPECNT AS**

**BEGIN**

**SELECT**

\*

**FROM**

(

**SELECT**

DISPLAY\_NAME Name,

email,

phone,

msg\_type

**FROM**

sql301.USERS

**INNER** **JOIN** SQL301.GROUP\_USERS **ON** sql301.users.USER\_GUID = sql301.GROUP\_USERS.USR\_ID

**INNER** **JOIN** sql301.messages **on** sql301.GROUP\_USERS.**GROUP\_ID** = SQL301.MESSAGES.**GROUP\_ID**

) t PIVOT(

**COUNT**(msg\_type) **FOR** msg\_type **IN** (

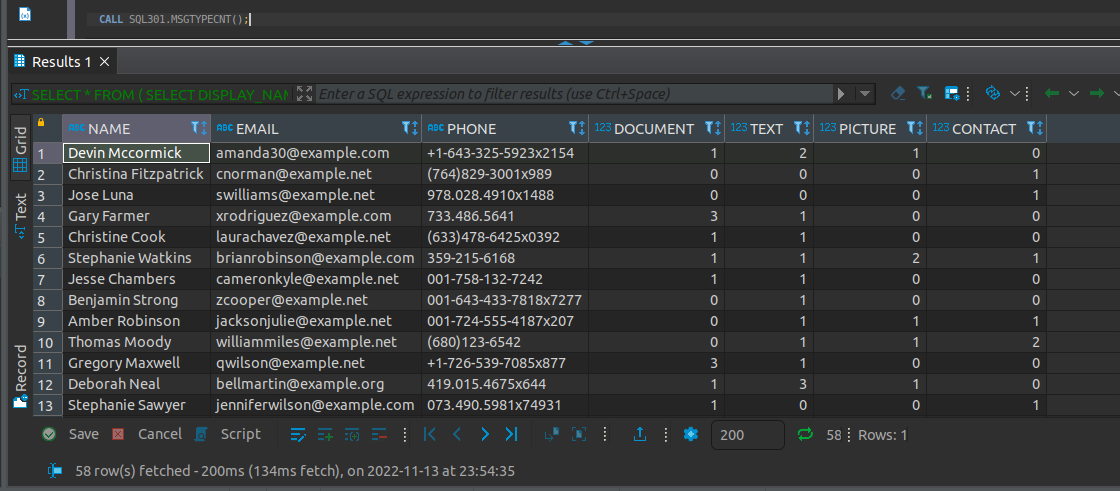
'document' Document, 'text' Text, 'picture' Picture,

'contact' Contact

)

);

**END**



**Procedure 2:**

--probable suspious users

**CRATE PROCEDURE SQL301.SUSPUSER AS**

**BEGIN**

**SELECT**

u.DISPLAY\_NAME,

u.EMAIL,

u.PHONE,

**count**(al.IP\_ADDR),

(

**CASE** **WHEN** **count**(al.IP\_ADDR)>= 2 **THEN** 1 **ELSE** 0 **end**

) is\_suspious

**FROM**

sql301.users u

**INNER** **JOIN** sql301.ACCESS\_LOG al **ON** al.USER\_ID = u.USER\_GUID

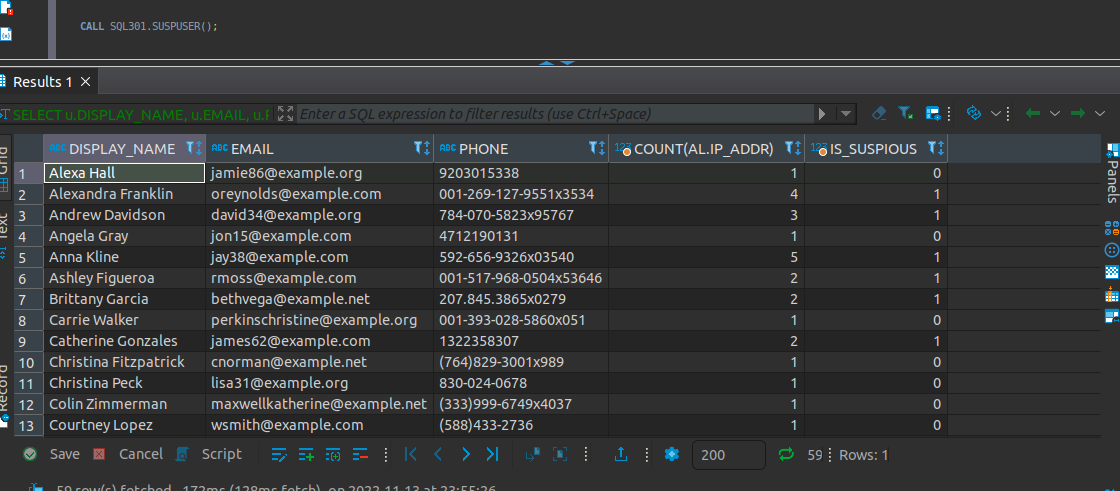
**GROUP** **BY**

u.DISPLAY\_NAME , u.EMAIL ,u.PHONE

**ORDER** **BY**

u.DISPLAY\_NAME

**END**



**Procedure 3:**

--USER WITH maximum message

**CRATE PROCEDURE SQL301.MAXMSG AS**

**BEGIN**

**SELECT**

u.DISPLAY\_NAME,

**count**(u.DISPLAY\_NAME)

**FROM**

sql301.USERS u

**INNER** **JOIN** sql301.MESSAGES m **ON** u.USER\_GUID = m.USER\_ID

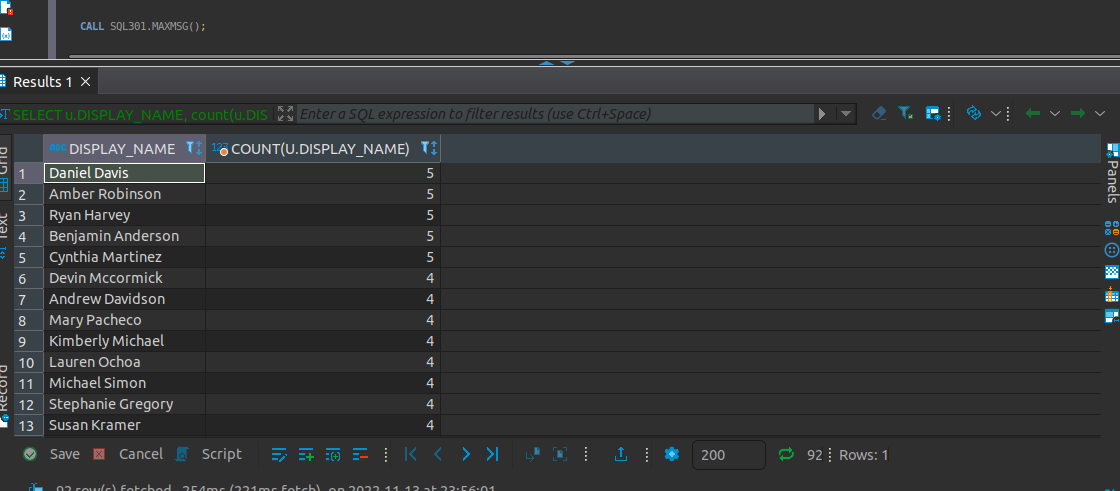
**GROUP** **BY**

u.DISPLAY\_NAME

**ORDER** **BY**

**count**(DISPLAY\_NAME) **desc**

**END**

****

**Part- 9 Database administration scripts**

The simple SQL script shows the table and column metadata are examples of Database administration scripts. The database system catalog includes metadata tables and views that describe the implementations of individual schemas.

Below are two different views of DBA scripts used for USERSS table:

a) List Integrity constraints

b) List Index structure

**a) List integrity constraints**

Below query gives us the list of all constraints which are applied to particular in summary format. Integrity constraints captures the business rules and protect the quality of data. Integrity constraints can be enabled or disabled at any time.

Graphical user interface, text, application, email

Description automatically generated

**b) List Index structure**

A table can have as many as indexes that improve performance. Monitoring the state and usage of the index structures is an ongoing administrative task.

Graphical user interface, text, application, email

Description automatically generated

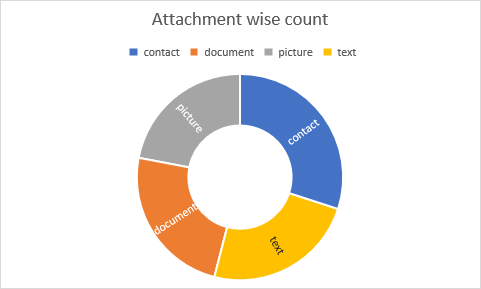
**Part- 10 Data Visualization**

Below are the data visualization for various tables in Chatting Application dataset:

Initially, we have applied Partitioning technique to USERS table for “CREATED\_AT” Column. Below are the six different partitions. For all partitions, we have count of users. Below is the Bar chart for decade wise active user count.

As a next step, it is important to find how many users in the table are active and in-active over the last few decades. Below is the Pie chart for Active and In-active users.

In group chats, users prefer to send the various attachments for sharing purposes. Below are attachment types are sent by various users in group chats.



**Part- 11 Conclusion**

In conclusion:

* we have created a Chatting Application Oracle SQL database which comprises of 12 tables.
* Defined key constraints for all 12 tables along with attributes and constraint name.
* We have applied Normalization technique, to reduce the database redundancy. Also, broke contact table to Phone book and contact. Where Phone book stores ID and User ID. Contact stores ID, First Name, Last Name, Phone and Profile Pic.
* Applied Partitioning technique for users table, Resulting table will have series of active users over the years by decade.
* Written a few sample queries to check whether Database is giving right result. We can confirm that output as expected.
* Finally, we have summarized all the important metrics of dataset using data visualization.