Symbiosis Institute of Technology

A DBMS Project Report on

Reddit

Submitted by

Safeer Firoz Khan (Roll No 18070122033)

Ritvik Ghattuwar (Roll No 18070122051)

Priyanshu Meena (Roll No 18070122045)

Rahul Airan (Roll No 18070122047)

Under the Guidance of

(Prof. Shruti Patil)

Department of Computer Science

SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE

Index

Table of Contents

Introduction

The internet is a huge place with new users connecting every day. The word internet has been added to our necessities of life. One of the basic things people do is to communicate, give opinions, discuss with a new tool in their hands they can now talk to people from the end of the world. So naturally, we decided to choose the website that calls itself the front page of the internet-**Reddit**. Reddit is a website/app where people can post any stuff and then people can discuss it, saving you the trouble of finding various opinions and even finding answers to some trivial questions. It's an incredible source of information on pretty much every field of human knowledge. Today Reddit is so large that there are more than **430 million monthly active Reddit users worldwide**. In this project, we attempt to implement the database that Reddit maintains in all its complexity.

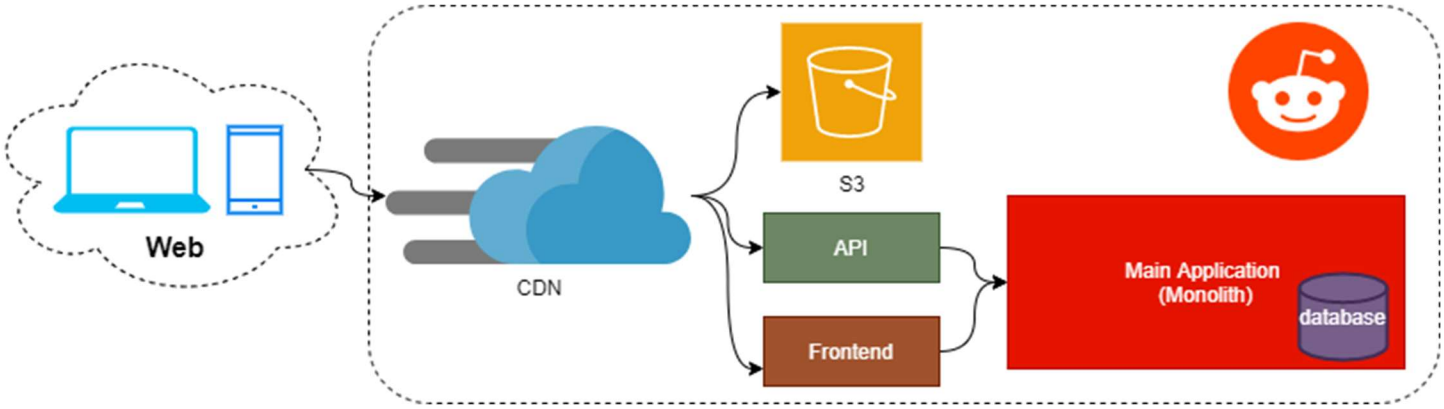
Problem Statement

The aim of this project is to implement the database used in Reddit website.This project is a prototype of the database used by the database enginners in Reddit.We have tried to show how Reddit database is managed through this project. This project has been implemented in MySQL.

Solution

We have created a well-defined database of Reddit in MySQL that is maintained by its developers for the website. The database will contain information about the info on which type of account user has, whether it is an advertising account and a normal account User. The basic info about the user and how one can vote, comment, create, browse a contribution (post and comment), the info regarding chat rooms the user joins, communities the user creates and follows, the followers, messages, the awards he can give. It also contains info about if a user has reported anything. The users which will have premium accounts. It also contains info of advertisement accounts, their campaigns, ad groups, and ads.

Architecture



The System design of Reddit includes the following modules:

1. **Web Client**
   The web client can be the Reddit website or the Reddit mobile application.
2. **CDN (Content Delivery Network)**
   Since Reddit serves over a million users, it is important to make the experience seamless. CDN allows content to be cached closer to the users, maintains and avoids site crashes due to high traffic, and avoids DDOs attacks.
3. **API**
   The API is the endpoint that serves the client with REST. It connects to the Main Application and returns the data in JSON or XML to the client application. Besides it also provides an open API for third-party applications
4. **Frontend (server-side render)**
   Reddit Web browser application is a server-side rendered application. This module is responsible for server-side rendering for the frontend.
5. **S3 (Bucket storage)**
   All Media links uploaded to Reddit directly are stored in AWS S3 bucket storage. The bucket stores blobs or files and their respective links are maintained in the main database.
6. **Main Application(Monolith)**
   This consists of all the business logic of Reddit. It's a monolith design application that connects to the database. To serve over a million users at times, there are several instances of this application running, and hence there may be load balancing and auto-scaling also present here.
7. **Database**
   This is the database we are designing. It connects to the monolith application where the business logic is implemented.

Functionalities

- One can create a normal user account or advertisement account.
- Each post belongs to a community
- Each post can have upvotes, downvotes, and comments
- Each post can have several awards
- Each comment can have another comment as a reply
- Posts can have optional tags
- Each post or comment can be saved by a user.
- User can create a profile (login and sign up, username).
- User can have followers.
- User can follow communities or users.
- User can post to a community.
- User can comment several times to any post or comment.
- User can upvote or downvote any post or comment, but only once per post/comment.
- User can be a moderator of one or more communities.
- User can report a post or comment.
- User can purchase premium subscriptions.
- User can direct message with other Users
- User can create or join chat rooms.
- User can purchase an Award that can be given from the user to another user for a post or comment.
- Awards can be of several types depending on image and price.
- Community can be created by the user.
- Community has at least one moderator.
- Community has users as followers
- Community has a unique name
- Community has a description and specific links and about sections.
- Community consists of Post
- Each Post can be a media or some text, preceded by a Title.
- Each post has a comment associated with it.
- Posts can have tags
- Posts can have votes by user
- Each comment is associated with a post.
- A comment can be a reply to another comment in the same post.
- Each comment can also receive votes and awards.
- Each post or comment can be reported by a user.
- Each report is reviewed by the Reddit admins.
- Admin will manage reports against a post or comment sent by the user.
- A separate account is required for advertisements.
- Advertisement User account can create a campaign

- Campaign will have an objective. Objective decides the cost of the campaign.
- Ad Group consists of Ad posts, and determines the location and schedule of advertisements in that group.

Entities

1. User
2. Post
3. Community
4. About Section Entry
5. Comment
6. Vote
7. Tag
8. Award Transaction
9. Award Type
10. Chat room
11. Subscription
12. Report
13. Advertisement account
14. Campaign
15. Ad Group
16. Ad post
17. Objective
18. Admin
19. Account
20. Contribution

Entities and Relationships

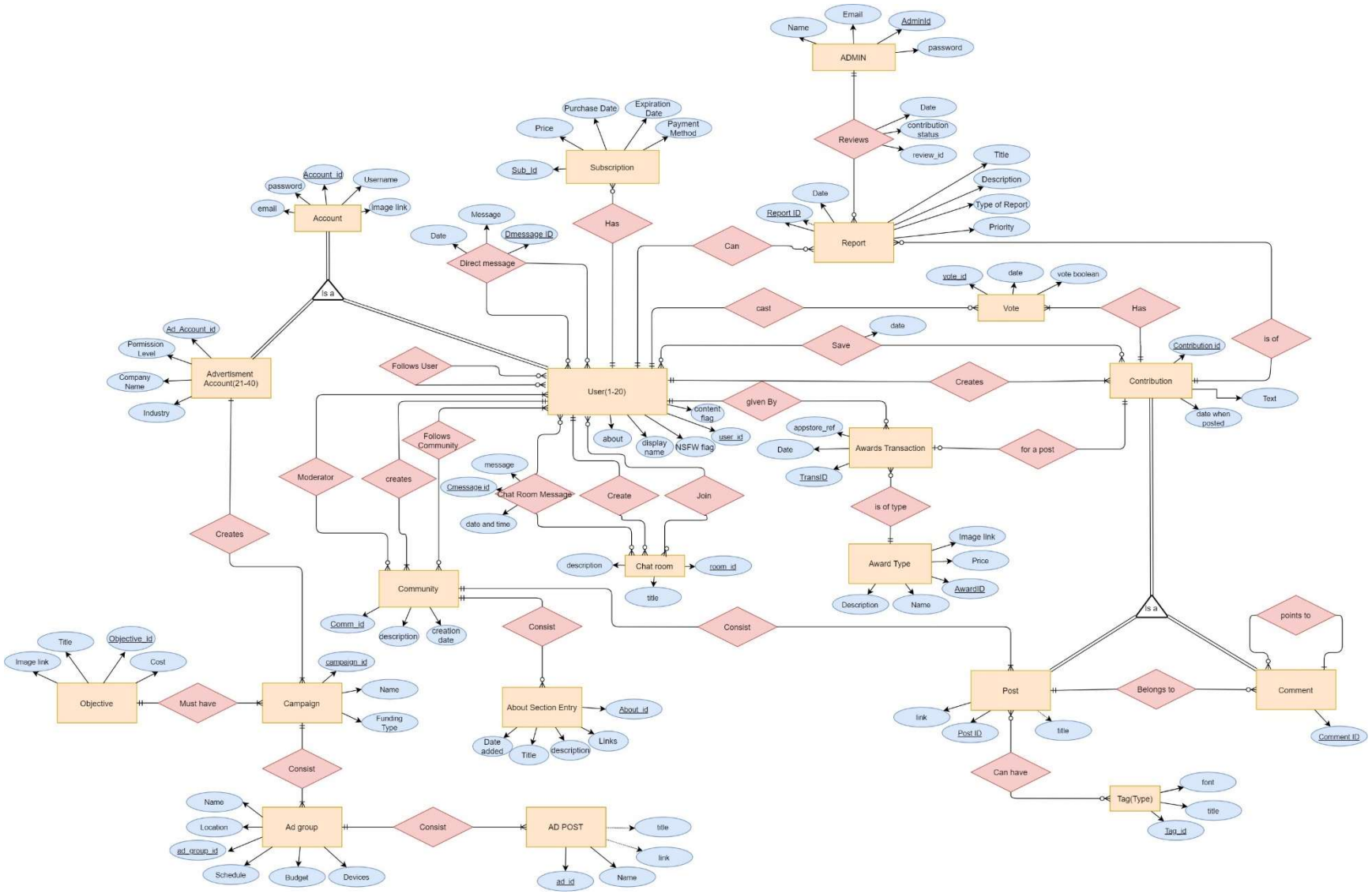
Entity	Relationship	Entity
User	User can post to a community (One to Many Relation)	**Contribution**
User	User can follow many communities and a community can have many followers. (Many to Many Relation)	**Community**
User	User can report any post/comment to Admin (One to Many Relation)	**Report**
User	User can subscribe to a premium account many times (One to many relation)	**Subscription**
User	User can message in a Chat Room and a chat room can contain messages of many users (Many to Many relation)	**Chat Room**
User	User can give awards to other users for any post or a comment (One to Many relation)	**Award Transaction**
User	User can cast a vote to many Contributions (Post/Comment) (One to Many relation)	**Vote**
User	User can personally message and receive from other Users (Many to Many relation)	**User**
User	User can follow many other Users and can be followed by them (Many to Many relation)	**User**

User	Many Users can be moderators to a Community, one User can be a moderator of many communities (Many to Many relation)	**Community**
User	User can save Posts/Comments and a particular contribution can be saved by many users (Many to Many relation)	**Contribution**
User	User can create a Chat Room (One to Many relation)	**Chat Room**
User	Many users can join many Chat Rooms (Many to Many)	**Chat Room**
Report	Any Post/Comment can be reported as many times (One to Many relation)	**Contribution**
Report	Many reports can be monitored by Admin (Many to One relation)	**Admin**
Community	A Community consists of About Section Entry	**About Section Entry**
Award Transaction	Each transaction will involve a particular Award Type (Many to One relation)	**Award Type**
Award Transaction	Each transaction will involve a Post/Comment (One to One relation)	**Contribution**
Post	A Post can have many Tags (Many to Many relation)	**Tag Type**
Post	Many Posts can belong to a Community (Many to One relation)	**Community**
Ad_post	Many Ads may belong to an Ad Group	**Ad Group**

	(Many to One relation)	
Comment	A sub-comment will point to its parent Comment (Many to One relation)	**Comment**
Comment	Many Comments can belong to one Post (Many to One relation)	**Post**
Campaign	Many Campaigns can be created by an Advertisement Account (Many to One relation)	**Advertisement Account**
Ad Group	Many Ad Groups can belong to a Campaign (Many to One relation)	**Campaign**
Objective	A campaign will have an Objective (One to Many relation)	**Campaign**

EER Diagram

https://drive.google.com/file/d/1JYCB2KLQBlW27FslwgEq_qJitmCld16e/view?u



Relational Schema

Dotted Underline- foreign key
Underlined-Primary key

1. User

User_id	About	Display_name	NSFW_flag	account_id

Primary key:-User_id
Foreign key:- account_id
Candidate key :- User_id, display_name
Alternate key:- display_name

2. Follows_User

Follow_id	user_id_follower	user_id

Primary key:- Follow_id
Foreign key:- user_id_follower , user_id
Candidate key :- Follow_id
Alternate key:- No alternate key

3. Moderator

moderator_id	user_id	community_id

Primary key:- moderator_id
Foreign key:- user_id , community_id
Candidate key:- Moderator_id
Alternate key:- No alternate key

4. Post

post_id	Title	Link	community_id	contribution_id

Primary key :- post_id

Foreign key:- community_id , tag_id , contribution_id
Candidate key :- Post_id
Alternate key:- No alternate key

5. Community

Community_id	name	Description	Creation_date	user_id

Primary key:- comm_id
Foreign key:- user_id
Candidate key:- Comm_id, name
Alternate key:- name

6. Follows_Community

follows_community_id	community_id	user_id

Primary key:- follows_community_id
Foreign key :- community_id , user_id
Candidate key :- follows_community_id
Alternate key:- No alternate key

7. About Section Entry

About_id	Date_added	Title	Description	Links	community_id

Primary key :- about_id
Foreign key :- community_id
Candidate key :- About_id
Alternate key:- No alternate key

8. Comment

Comment_id	post_id	points_id	contribution_id

Primary key:- comment_id
Foreign key:- points_id, contribution_id,post_id
Candidate key :- comment_id

Alternate key:- No alternate key

9. Tag

tag id	Title	Font

Primary key:- tag_id
Foreign key:- no foreign key
Candidate key: tag_id, title
Alternate key:- Title

10. Post_Can_Have_Tags

post_tags_id	post_id	tag_id

Primary key:- post_tags_id
Foreign key:- post_id, tag_id
Candidate key :- post_tags_id
Alternate key:- No alternate key

11. Award Transaction

transaction_id	Date	appstore_ref	user_id	contribution_id	award_id

Primary key:- transaction_id
Foreign key:- user_id, contribution_id, award_id
Candidate key:- transaction_id
Alternate key:- No alternate key

12. Award Type

award_id	Name	Description	Price	Image_link

Primary key:- award_id
Foreign key:- No foreign key
Candidate key:- award_id, name, image_link
Alternate key:- Name, Image_link

13. Chat room

room_id	Title	Description	user_id

Primary key:- room_id
Foreign key:- user_id
Candidate key:- room_id
Alternate key:- No alternate key

14. Join_chatroom

Join_id	user_id	room_id

Primary key:- Join_id
Foreign key:- user_id, room_id
Candidate key:- Join_id
Alternate key:- No alternate key

15. Subscription

sub_id	Price	Purchase_date	Expiration_date	Payment_date	user_id

Primary key: sub_id
Foreign key: user_id
Candidate key:- sub_id
Alternate key:- No alternate key

16. Report

report_id	Date	Title	Description	Report_type	Priorty	user_id	contribution_id

Primary key:- report_id
Foreign key:- user_id, contribution_id
Candidate key :- report_id
Alternate key:- No alternate key

17. Advertisement account

ad_account_id	Permission_level	Company_Name	Industry	account_id

Primary key:- ad_account_id
Foreign key:- account_id
Candidate key :- ad_account_id
Alternate key:- No alternate key

18. Campaign

campaign_id	Name	Funding_Type	ad_account_id	objective_id

Primary key:- campaign_id
Foreign key:- ad_account_id, objective_id
Candidate key:- campaign_id
Alternate key:- No alternate key

19. Ad Group

ad_group_id	Name	Location	Schedule	Budget	Devices	campaign_id

Primary key:- ad_group_id
Foreign key:- campaign_id
Candidate key :- ad_group_id
Alternate key:- No alternate key

20. Ad Post

ad_id	name	title	link	ad_group_id

Primary key :- ad_id
Foreign key :- ad_group_id
Candidate key :- ad_id
Alternate key:- No alternate key

21. Objective

objective_id	Cost_type	Objective_type

Primary key:- objective_id
Foreign key:- No foreign key
Candidate key:- objective_id
Alternate key:- No alternate key

22. Admin

admin_id	name	email	password

Primary key:- admin_id
Foreign key:- No foreign key
Candidate key:- admin_id, email
Alternate key:- email

23. Account

account_id	email	username	password	image_link

Primary key:- account_id
Foreign key:- No foreign key
Candidate key:- account_id, email, username
Alternate key:- email, username

24. Contribution

contribution id	text	date_posted	user_id

Primary key :- contribution_id
Foreign key :- vote_id , user_id
Candidate key:- contribution_id,
Alternate key:- No alternate key

25. Vote

vote_id	vote_date	vote_boolean	user_id	contribution_id

Primary key:- vote_id
Foreign key :- user_id , contribution_id
Candidate key:- Vote_id
Alternate key:- No alternate key

26. Direct_Message

Dmessage_id	date	message	user_one	user_two

Primary key:- Dmessage_id
Foreign key :- user_one , user_two
Candidate key:- Dmessage_id
Alternate key:- No alternate key

27. Chat Room Message

Cmessage_id	Message	Date_and_time	room_id	user_id

Primary key:- Cmessage_id
Foreign key :- room_id , user_id
Candidate key:-Cmessage_id
Alternate key:- No alternate key

28. Save

save_id	date	user_id	contribution_id

Primary key :-save_id
Foreign key :- Contribution_id , user_id
Candidate key :- save_id
Alternate key:- No alternate key

29. Reviews

review_id	date	Contribution_removed	admin_id	report_id

Primary key:-review_id
Foreign key:- admin_id, report_id
Candidate key:- review_id
Alternate key:- No alternate key

Codd's Rules

Dr. Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied to any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule
The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

- This rule is **followed** in our database. The database is implemented in the table format only in MySQL (Relational Database Management system).

Rule 2: Guaranteed Access Rule
Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

- This rule is **followed** in our database. Every data value can be accessed logically using queries in MySQL.

Rule 3: Systematic Treatment of NULL Values
The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

- This rule is **followed** in our database. The NULL value in the database implemented is data missing in our database implemented through MySQL.

Rule 4: Active Online Catalog
The structure description of the entire database must be stored in an online catalog, known as a data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

- This rule is **followed** in our database. We have the query language to check all the constraints and data in a database.

Rule 5: Comprehensive Data Sublanguage Rule
A database can only be accessed using a language having a linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

- This rule is **followed** in our database. MySQL software is used to implement this database in SQL.

Rule 6: View Updating Rule
All the views of a database, which can theoretically be updated, must also be updatable by the system.

- This rule is **not followed** in our database. Although updating the view will update the table used for creating it, it is not recommended by most of the database. Hence this rule is not used in most of the databases.

Rule 7: High-Level Insert, Update, and Delete Rule
A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection, and minus operations to yield sets of data records.

- This rule is **followed** in our database. High-level operations are allowed in our database and one single query can be used to update a set of records.

Rule 8: Physical Data Independence
The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

- This rule is **followed** in our database. Any change done would be absorbed by the mapping between the conceptual and internal levels in MySQL.

Rule 9: Logical Data Independence
The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rules to apply.

- This rule is **not followed** in our database. But in an ideal scenario, this is difficult to achieve since all the logical and user views will be tied so strongly that they will be almost the same.

Rule 10: Integrity Independence
A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

- This rule is **not followed** in our database. Because suppose we change the constraint to put a minimum value or change a range of any data then the application must also then show messages or errors accordingly. Although, primary key and foreign key constraints are maintained in our database.

Rule 11: Distribution Independence
The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

- This rule is **not followed** in our database. Since we are using the MySQL community edition, this condition is violated. For a distributed database, we will have to use MySQL cluster

Rule 12: Non-Subversion Rule
If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

- This rule is **followed** in our database. Users can only access data using SQL and no other lower-level language in MySQL. So the query written in the database by the user will be converted in the same low-level syntax and not any other language which will change the data integrity.

Considerations while designing

(conclusions of brainstorming) along with Justifications:

1. **Combining Post and Comment in 'is a' relationship as Contribution.**

   Post and Comment are related to several common entities. These relations are with entities such as User, Vote, Award Transaction, Report; and have the same cardinality as well. Besides, they even have several common attributes. Hence we decided to establish 'is a' relationship with a superclass as Contribution and have subclasses as Post and Comment. These would hence inherit all relationships and attributes of Contribution.

2. **Award Transaction relating to User and Contribution.**

   For the functionality of awards, there are two users and a post involved. However, as the post is already linked to one of the users through a relationship, there is only one explicit relationship between User and Award Transaction which is for the user that's giving the award. The user that is receiving the award is linked to the Contribution, hence we relate the contribution to the Award Transaction. Hence we get rid of a redundant relationship between the User and Award Transaction. Instead of two, we include only one.

3. **Vote as a separate entity rather than an attribute of Contribution.**

   For any social networking site, the votes(likes, hearts, whatever you may call it) decide the popularity of that post/contribution. Hence, there is a need for these votes to be legitimate and be able to be audited. If Vote were an attribute, it would only record the count of votes and not relate to the user that casts it, thereby allowing a user to vote multiple times the same vote, which is wrong. Since each User is entitled to vote a contribution only once, we should be able to trace that vote to the user, and hence maintain a record of the same. As we further investigate the Reddit architecture, there's a dedicated vote verification mechanism that would identify fraudulent votes and please the corresponding users. This would require Vote to be a separate entity of its own.

4. **Relationships with Comment Entity**

   Our initial consideration for Comment was to relate it to its superclass Contribution. This would result in the subclasses Post and Comment inheriting this relationship with comment. This approach would have every comment point to either a post or a comment. The issues we noticed here were that there is no direct relation with a comment which is a reply to another comment and the post. This would cause an issue as if the post is deleted, we would have to delete all comments one by one, instead of doing it with a single query; and we cannot maintain a count of total comments belonging to a post. Ultimately this may be a poor design choice and hence we chose an alternative approach.

   We explicitly define relationships between Post and Comment, and self-relation with Comment. This solves the above-mentioned issues. The main difference in doing this explicit relation versus the inherited relation is the difference in cardinality.

Anomalies,Functional Dependencies and Normalization

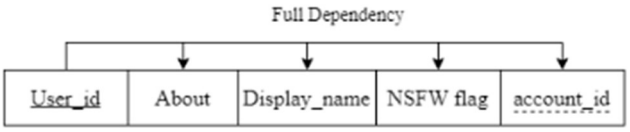
Dotted Underline- foreign key
Underlined-Primary key

1. User

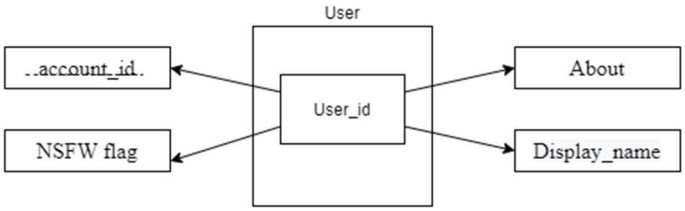
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

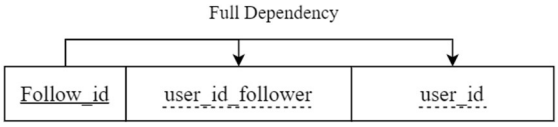
The above table is already normalized to BCNF(Boyce Codd Normal Form).

2. Follows_User

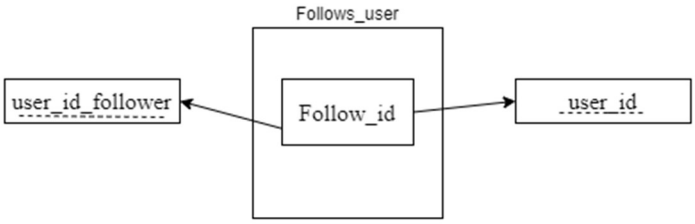
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

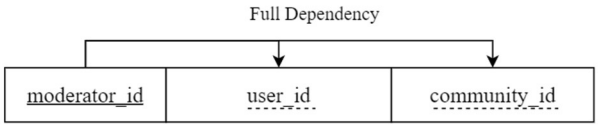
The above table is already normalized to BCNF(Boyce Codd Normal Form).

3. Moderator

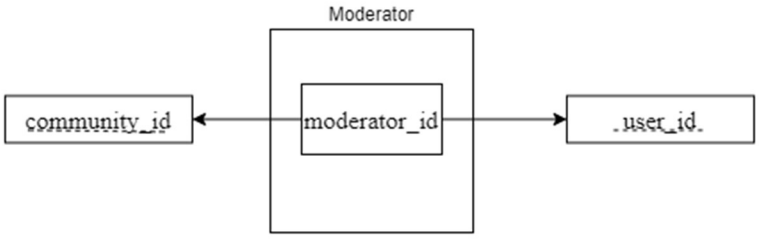
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

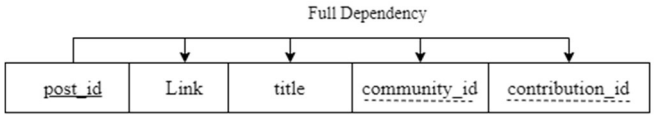
The above table is already normalized to BCNF(Boyce Codd Normal Form).

4. Post

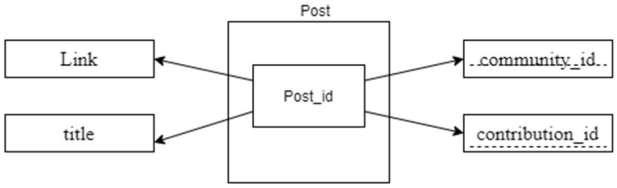
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

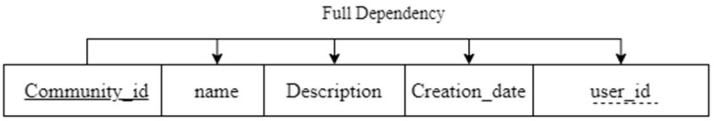
The above table is already normalized to BCNF(Boyce Codd Normal Form).

5. Community

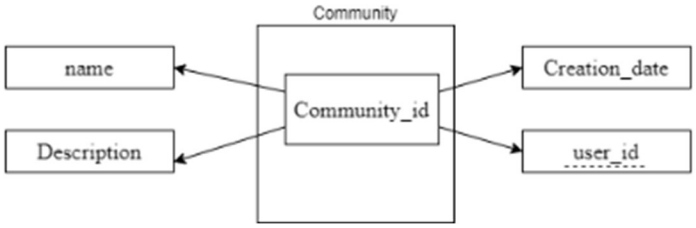
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

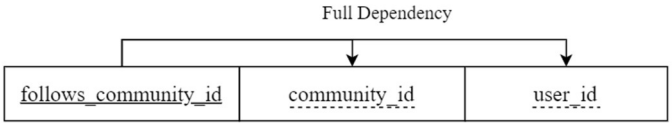
The above table is already normalized to BCNF(Boyce Codd Normal Form).

6. Follows_Community

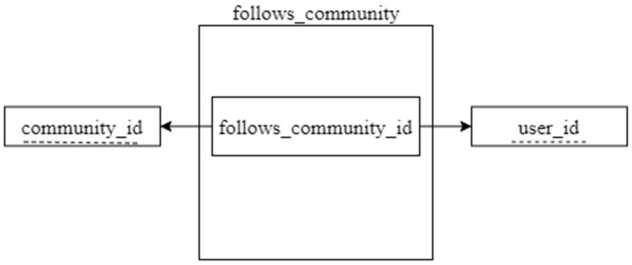
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

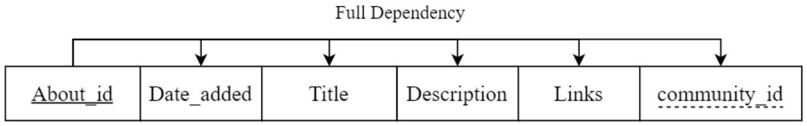
The above table is already normalized to BCNF(Boyce Codd Normal Form).

7. About Section Entry

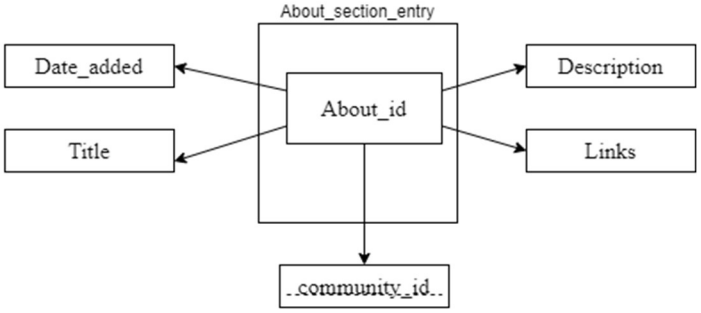
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

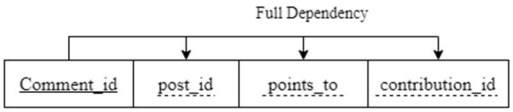
The above table is already normalized to BCNF(Boyce Codd Normal Form).

8. Comment

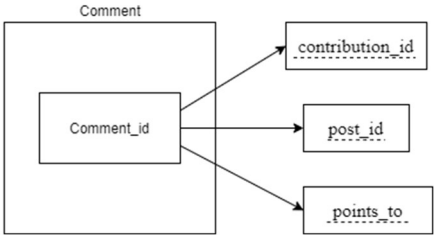
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

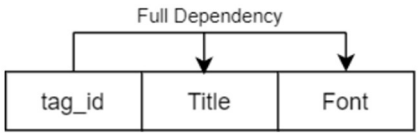
The above table is already normalized to BCNF(Boyce Codd Normal Form).

9. Tag

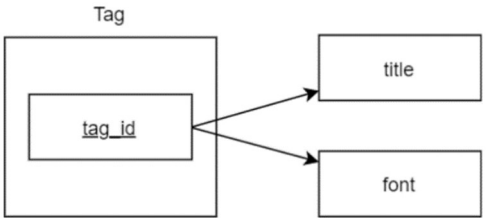
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

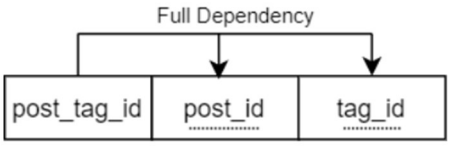


10. Posts_can_have_tags

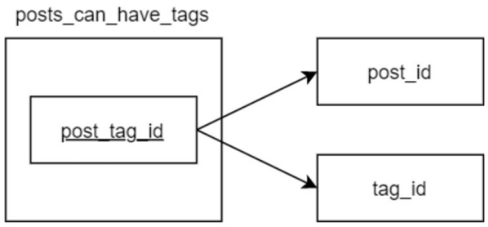
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

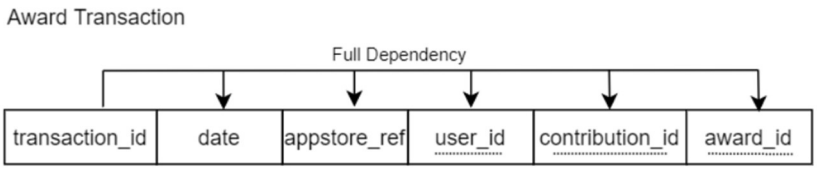


11. Award_transaction

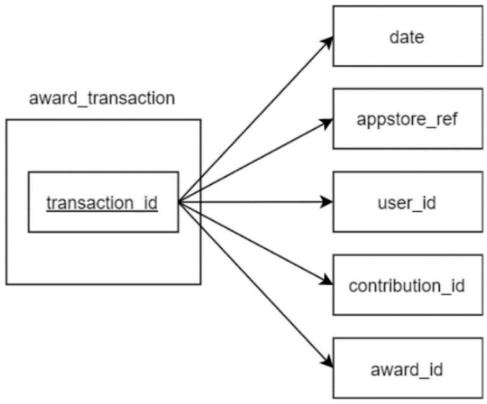
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

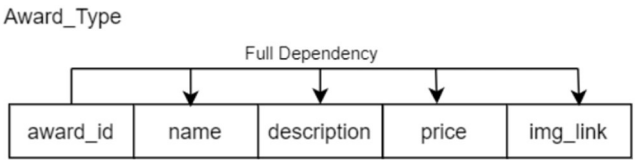


12. Award_type

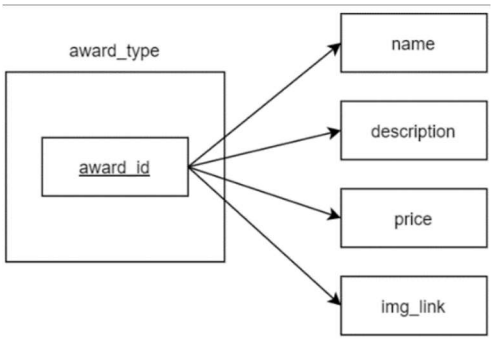
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

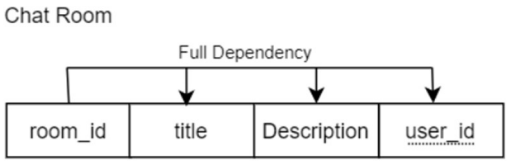


13. Chatroom

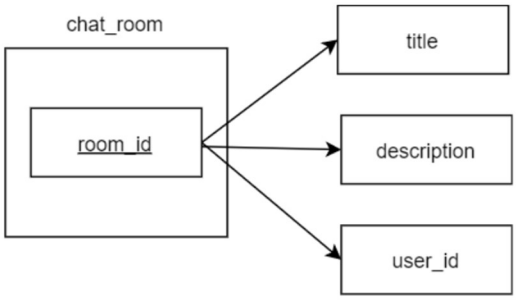
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

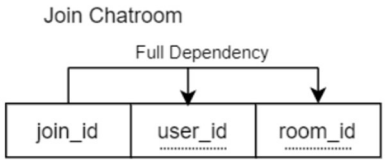


14. Join_chatroom

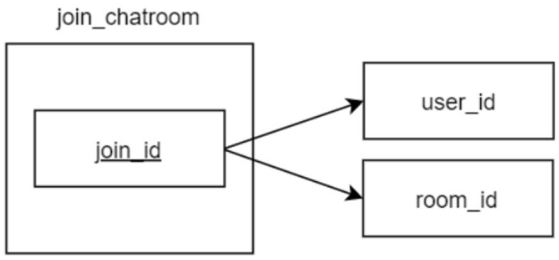
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



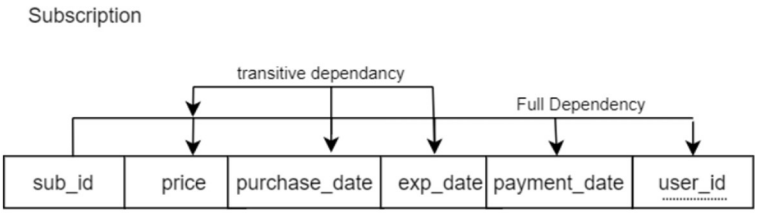
15. Subscription

Anomalies

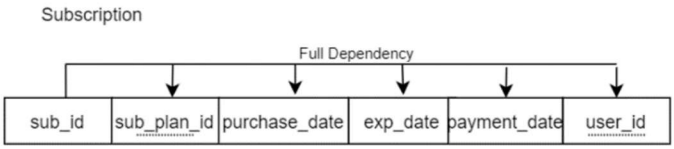
INSERTION ANOMALY	INSERTION ANOMALY PRESENT
DELETION ANOMALY	DELETION ANOMALY PRESENT
UPDATION ANOMOLY	UPDATION ANOMALY PRESENT

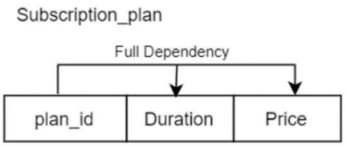
In the subscription table, we found out that anomalies existed due to transitive dependency between the attriibutes price, purchase_date and exp_date.
To get rid of these anomalies we have followed lossless decomposition of the above mentioned attributes into their separate tables. The following diagrams show the same.

Functional Dependencies

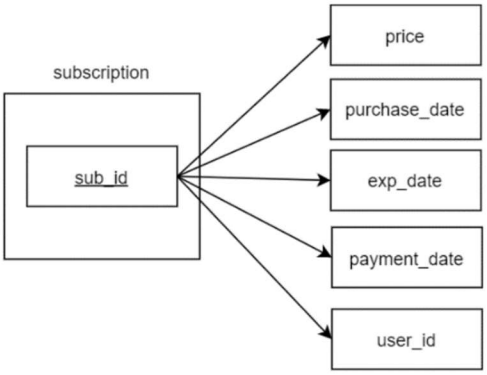


Decomposed into

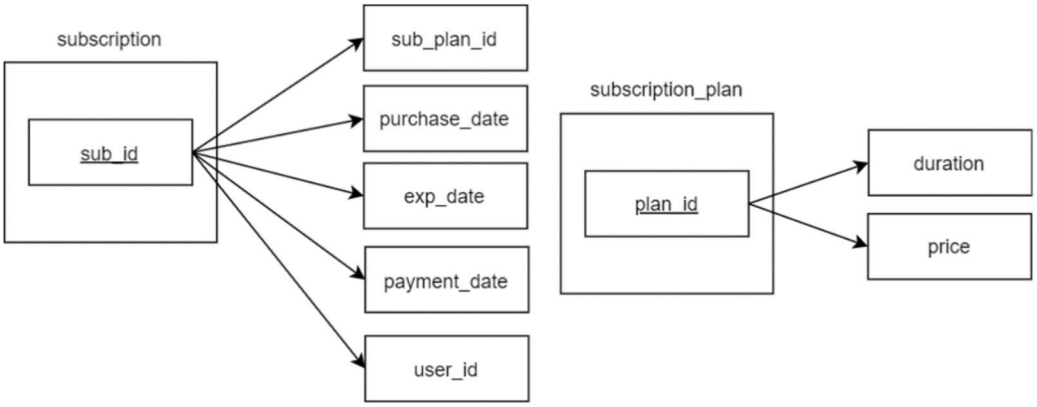




Functional Dependency Chart



Decomposed into



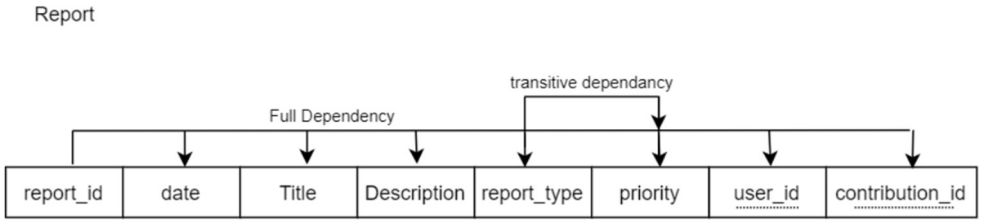
16. Report

Anomalies

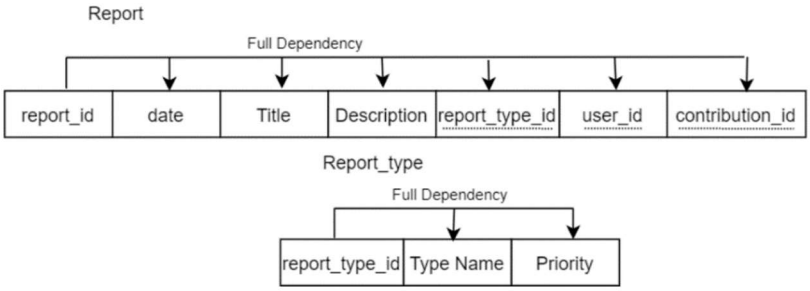
INSERTION ANOMALY	INSERTION ANOMALY PRESENT
DELETION ANOMALY	DELETION ANOMALY PRESENT
UPDATION ANOMOLY	UPDATION ANOMALY PRESENT

In the report table, we found out that anomalies existed due to transitive dependency between the attriibutes report_type and priority.
To get rid of these anomalies we have followed lossless decomposition of the above mentioned attributes into their separate tables. The following diagrams show the same.

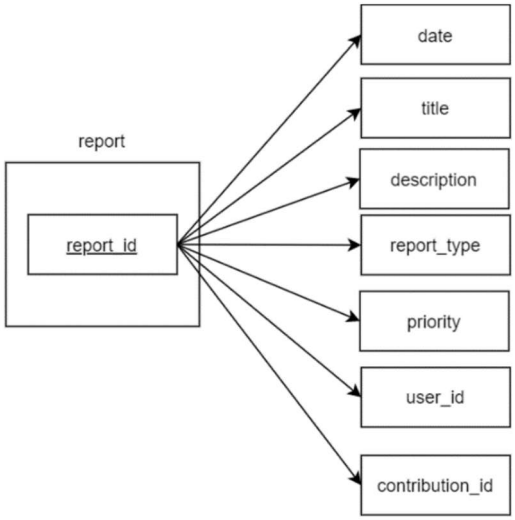
Functional Dependencies



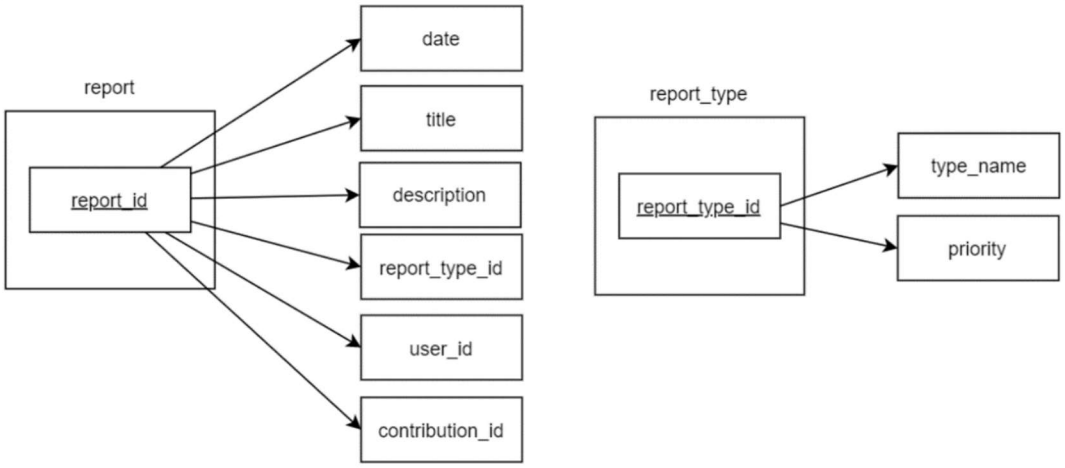
Decomposed into



Functional Dependency Chart



Decomposed into



17. Advertisement_account

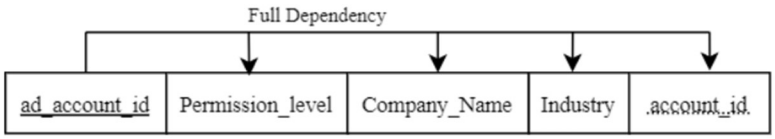
Anomalies

INSERTION ANOMALY	INSERTION ANOMALY PRESENT

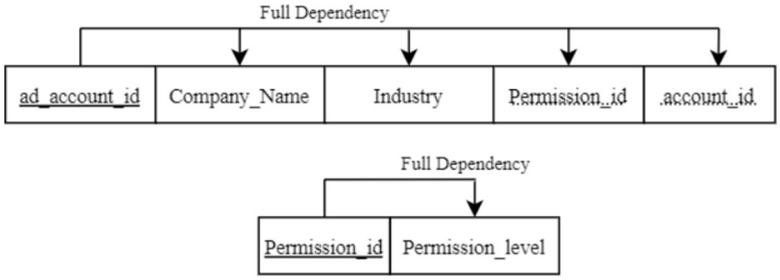
DELETION ANOMALY	DELETION ANOMALY PRESENT
UPDATION ANOMOLY	UPDATION ANOMALY PRESENT

In the advertisement_account, we found out that anomalies existed due to redundant data for the attribute, permission_level.
To get rid of these anomalies we have followed lossless decomposition of the above mentioned attribute into their separate tables. The following diagrams show the same.

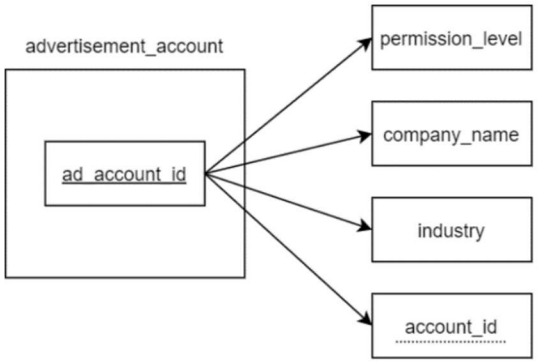
Functional Dependencies



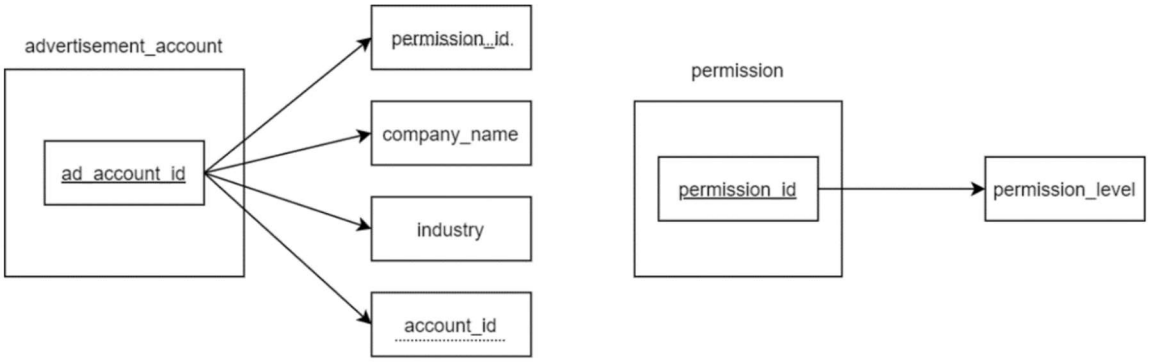
Decomposed into



Functional Dependency Chart



Decomposed into



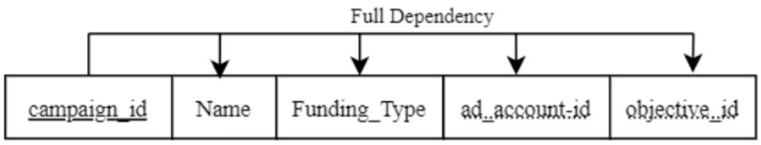
18. Campaign

Anomalies

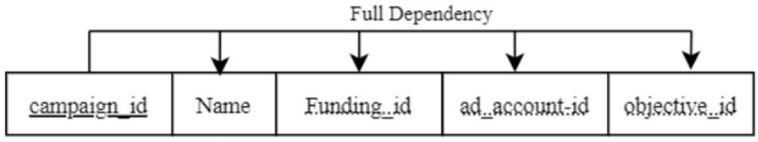
INSERTION ANOMALY	INSERTION ANOMALY PRESENT
DELETION ANOMALY	DELETION ANOMALY PRESENT
UPDATION ANOMOLY	UPDATION ANOMALY PRESENT

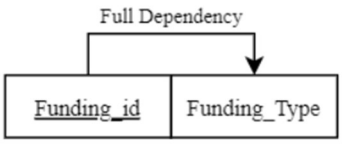
In the campaign table, we found out that anomalies existed due to redundant data for the attribute, funding_type.
To get rid of these anomalies we have followed lossless decomposition of the above mentioned attribute into their separate tables. The following diagrams show the same.

Functional Dependencies

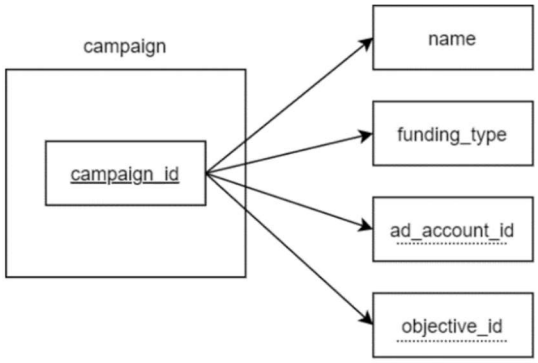


Decomposed into

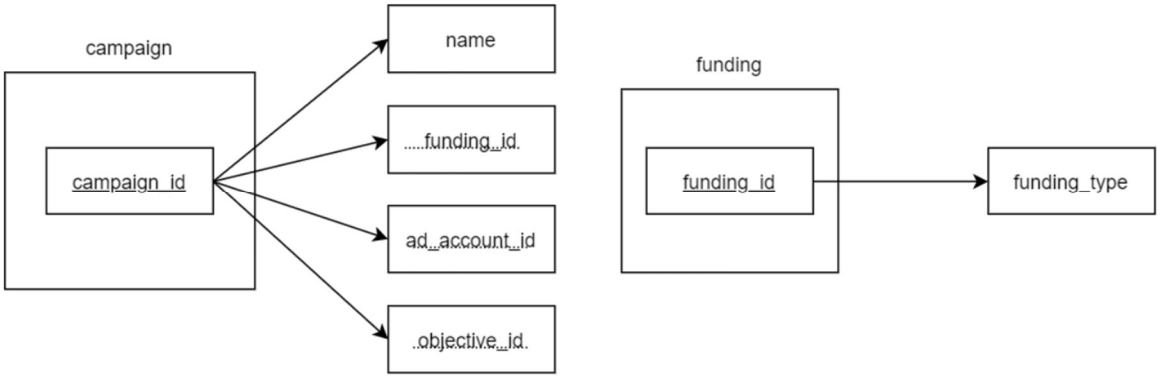




Functional Dependency Chart



Decomposed into



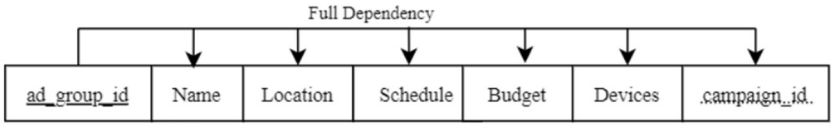
19. Ad_group

Anomalies

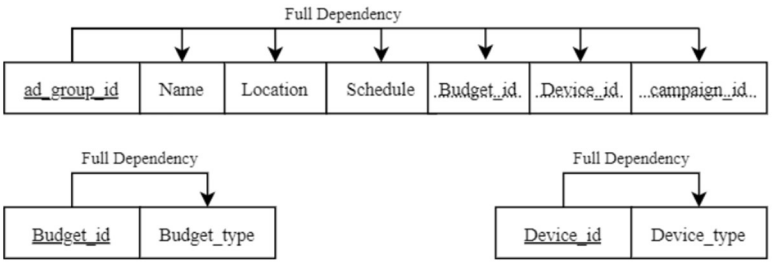
INSERTION ANOMALY	INSERTION ANOMALY PRESENT
DELETION ANOMALY	DELETION ANOMALY PRESENT
UPDATION ANOMOLY	UPDATION ANOMALY PRESENT

In the ad group table, we found out that anomalies existed due to redundant data for the attributes, budget and devices.
To get rid of these anomalies we have followed lossless decomposition of the above mentioned attributes into their separate tables. The following diagrams show the same.

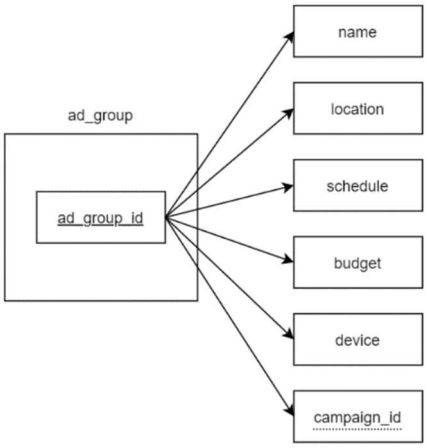
Functional Dependencies



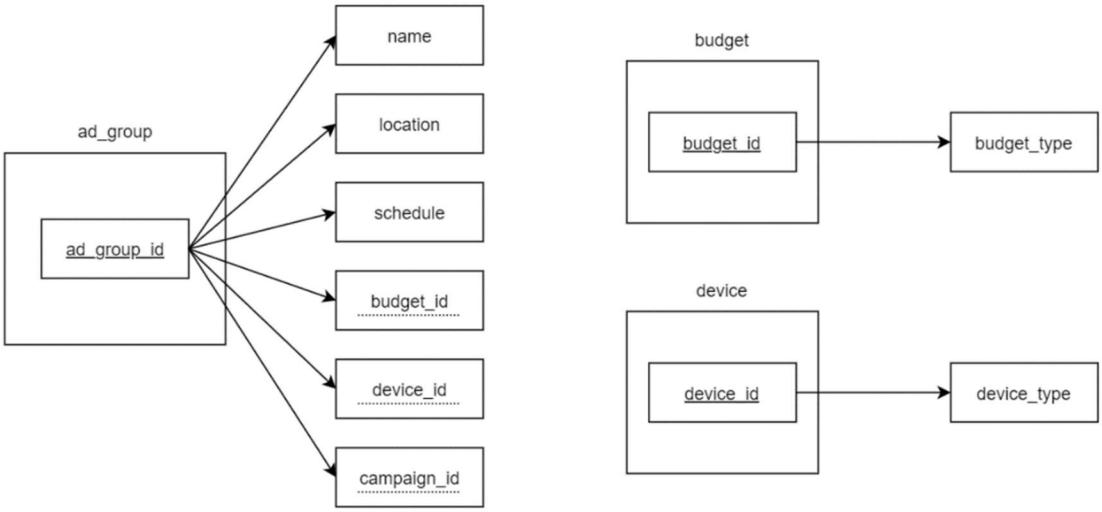
Decomposed into



Functional Dependency Chart



Decomposed into

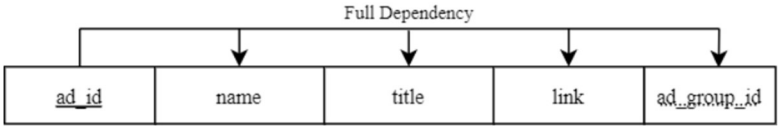


20. Ad_post

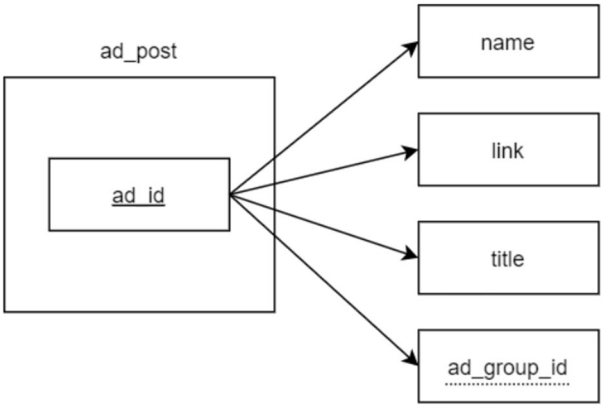
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

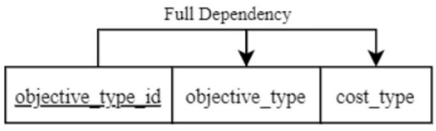


21. Objective

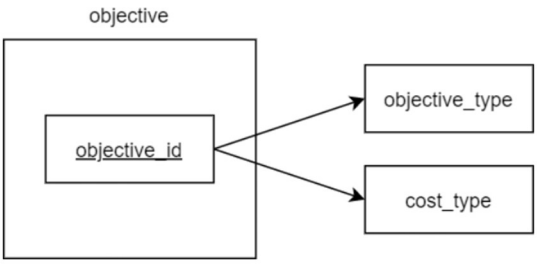
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart

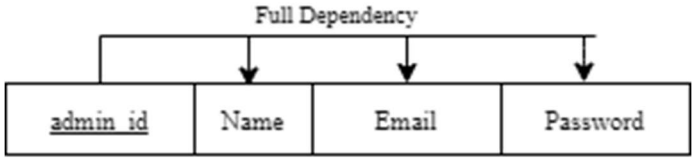


22. Admin

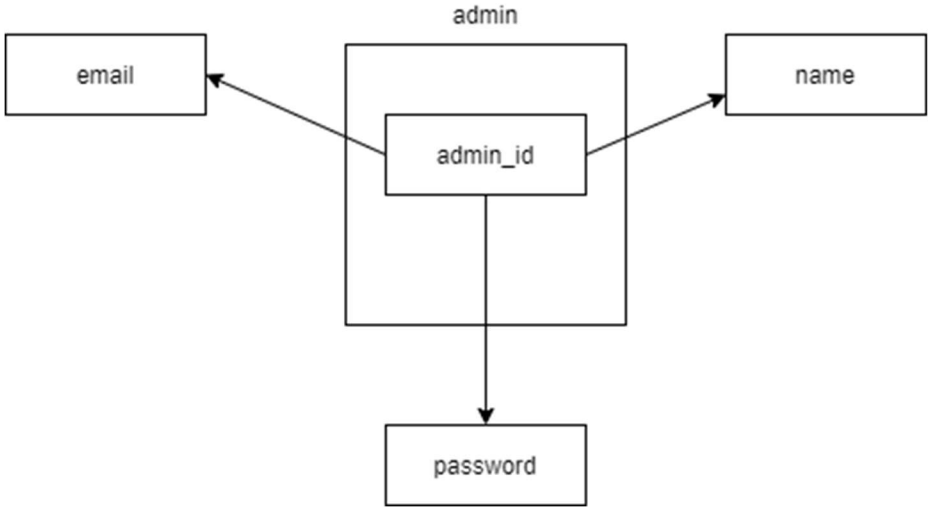
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

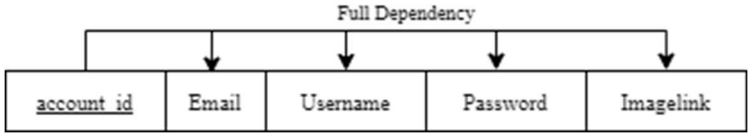
The above table is already normalized to BCNF(Boyce Codd Normal Form).

23. Account

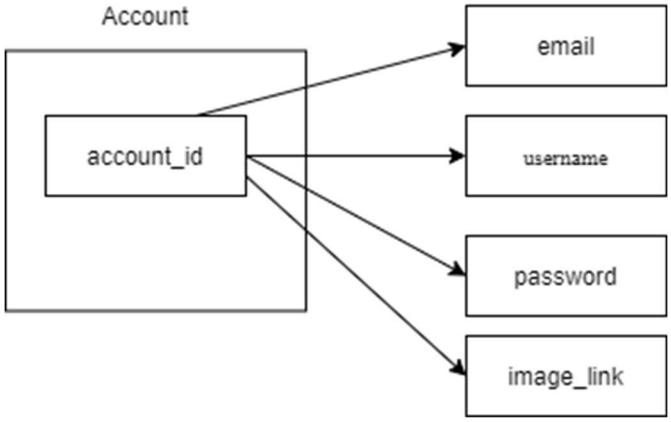
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

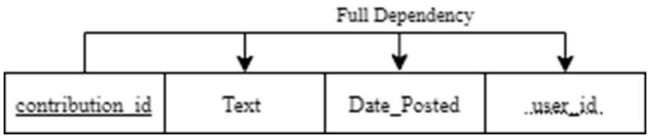
The above table is already normalized to BCNF(Boyce Codd Normal Form).

24. Contribution

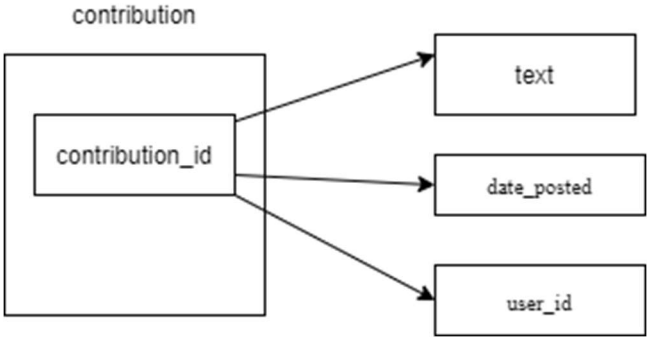
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

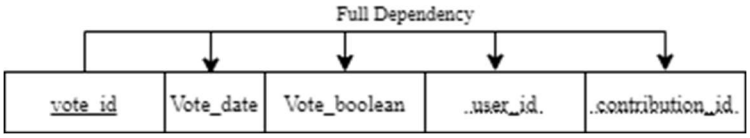
The above table is already normalized to BCNF(Boyce Codd Normal Form).

25. Vote

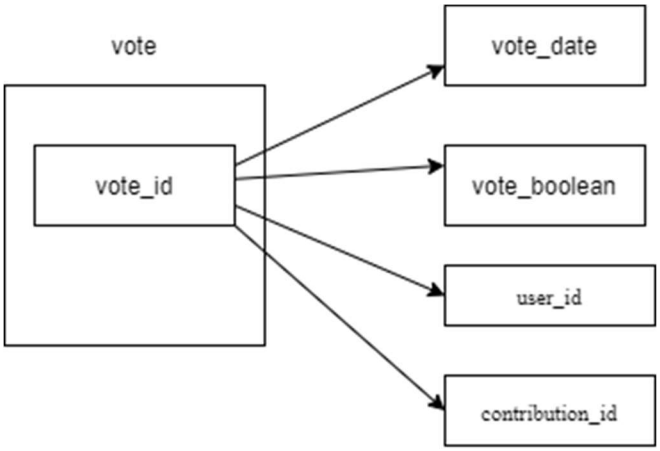
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

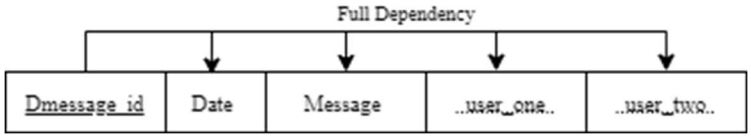
The above table is already normalized to BCNF(Boyce Codd Normal Form).

26. Direct Message

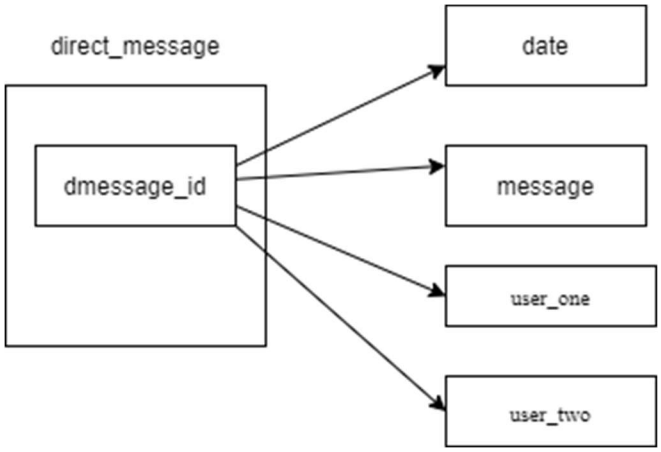
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

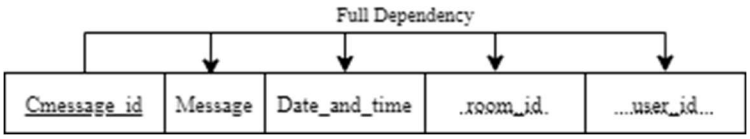
The above table is already normalized to BCNF(Boyce Codd Normal Form).

27. Chat Room Message

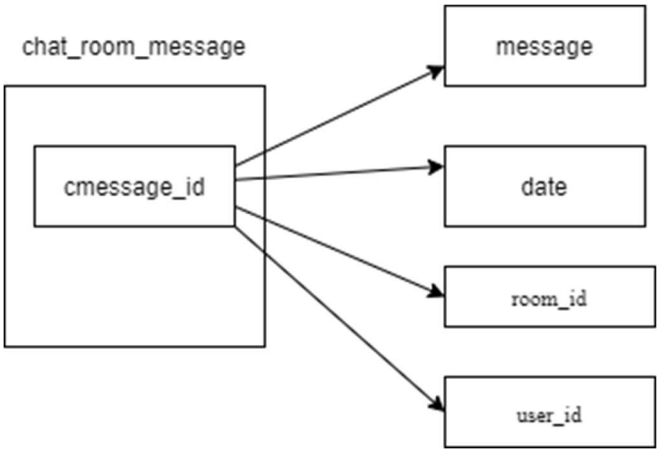
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

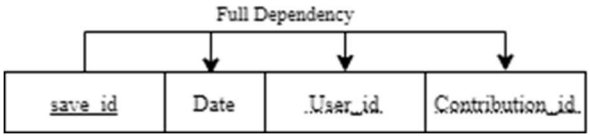
The above table is already normalized to BCNF(Boyce Codd Normal Form).

28. Save

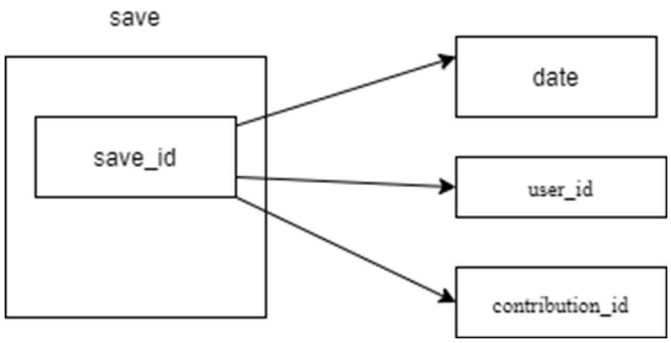
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

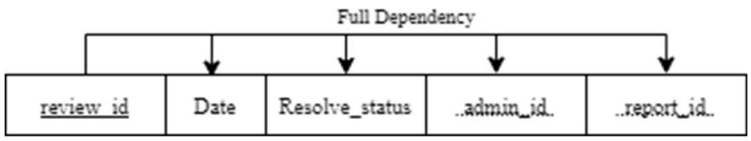
The above table is already normalized to BCNF(Boyce Codd Normal Form).

29. Reviews

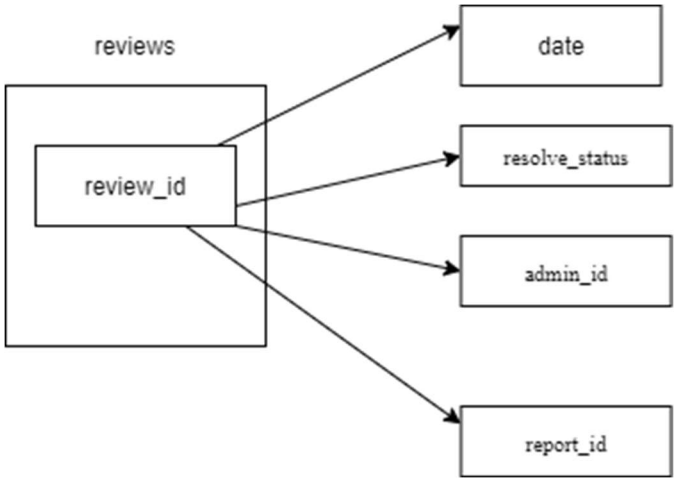
Anomalies

INSERTION ANOMALY	NO INSERTION ANOMALY PRESENT
DELETION ANOMALY	NO DELETION ANOMALY PRESENT
UPDATION ANOMOLY	NO UPDATION ANOMALY PRESENT

Functional Dependencies



Functional Dependency Chart



Normalization

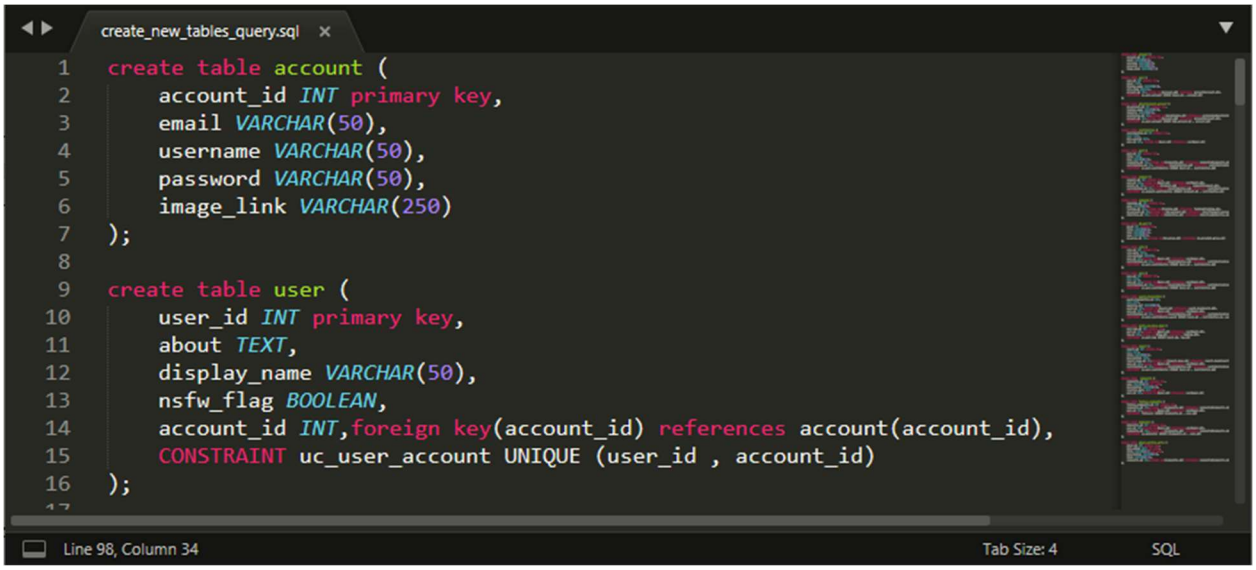
The above table is already normalized to BCNF(Boyce Codd Normal Form).

Implementation

As mentioned in the problem staement earlier, we implemenented the above relational schema on MySQL instances. We had initially hoped to use a cloud-based free mysql server, but unfortunately we had to switch to our local instance instead, due to restrictions on stored procedures, functions and triggers in free server providors

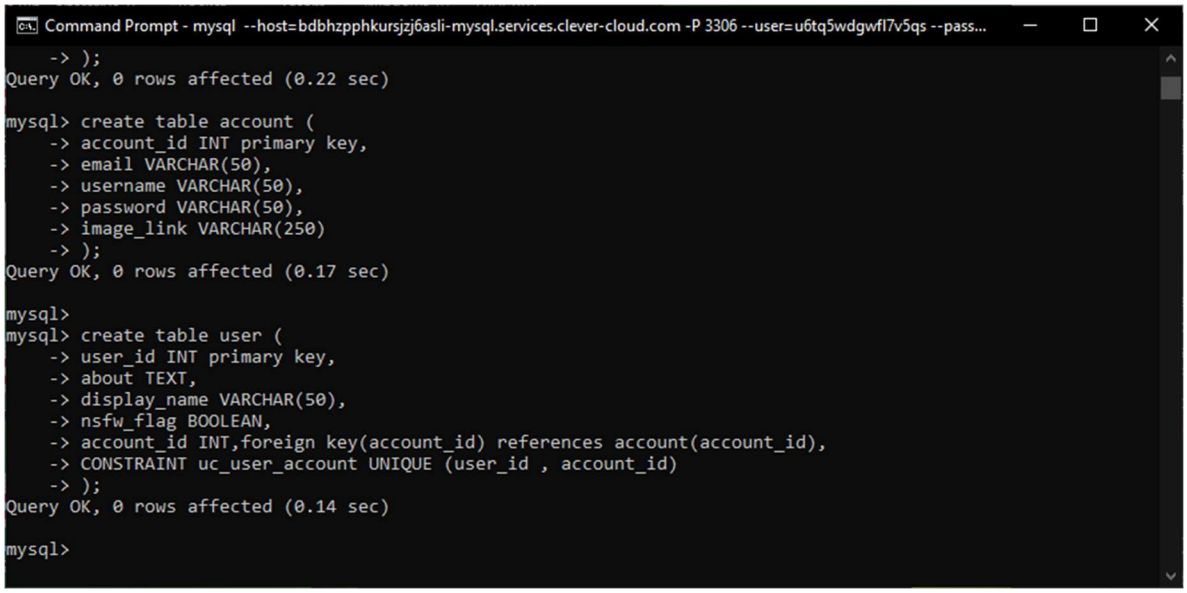
Creation of tables.

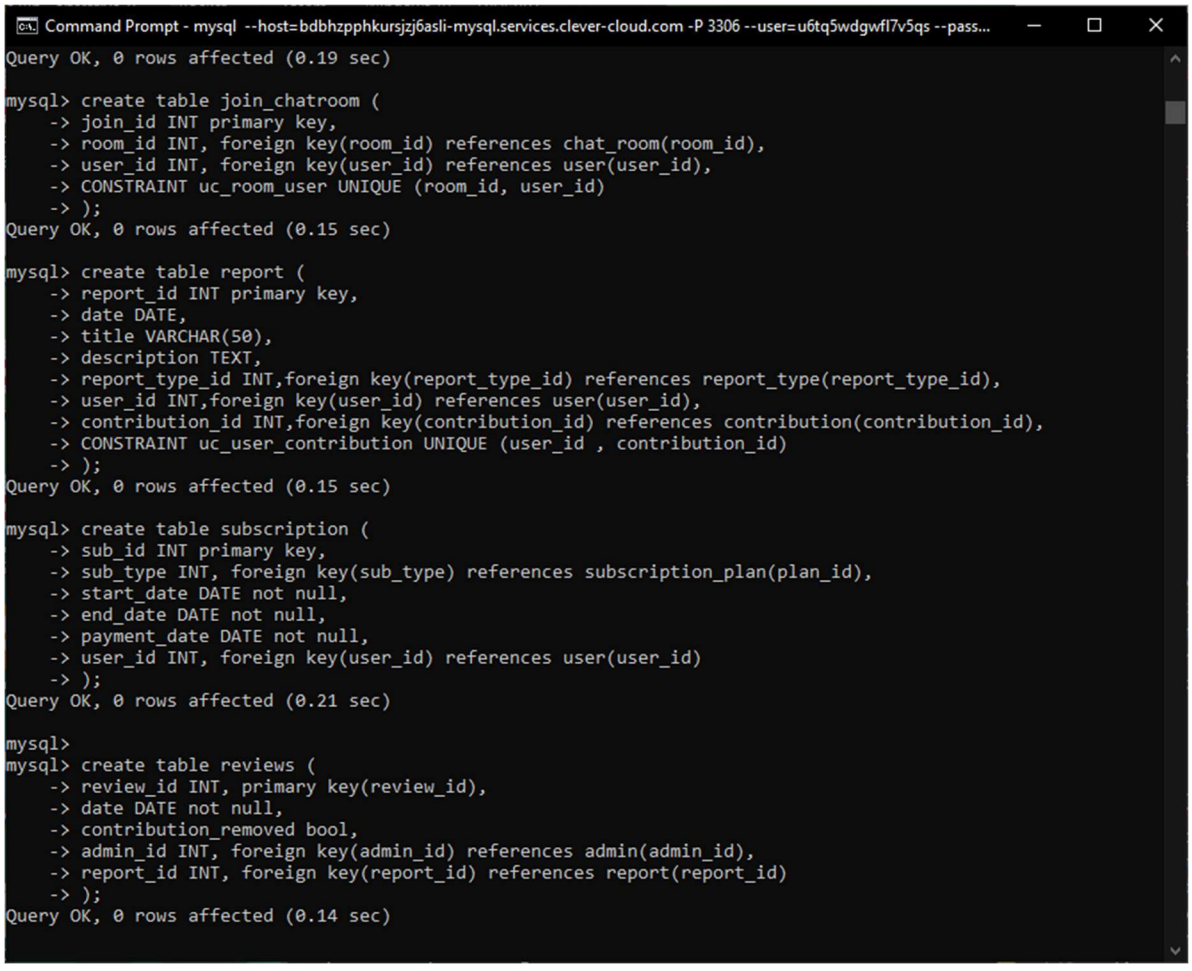
The create queries turned out to be a very crusial part in this project. After normalization, we had a total of 35 tables in our databes. For each of these tables, we had to write create queries. For each table, other than deciding upon the datatype of the attribute, we had to also decide upon the constraints, such as ‘not null’ and unique.
Below we have attached a screenshot of few of the create queries.



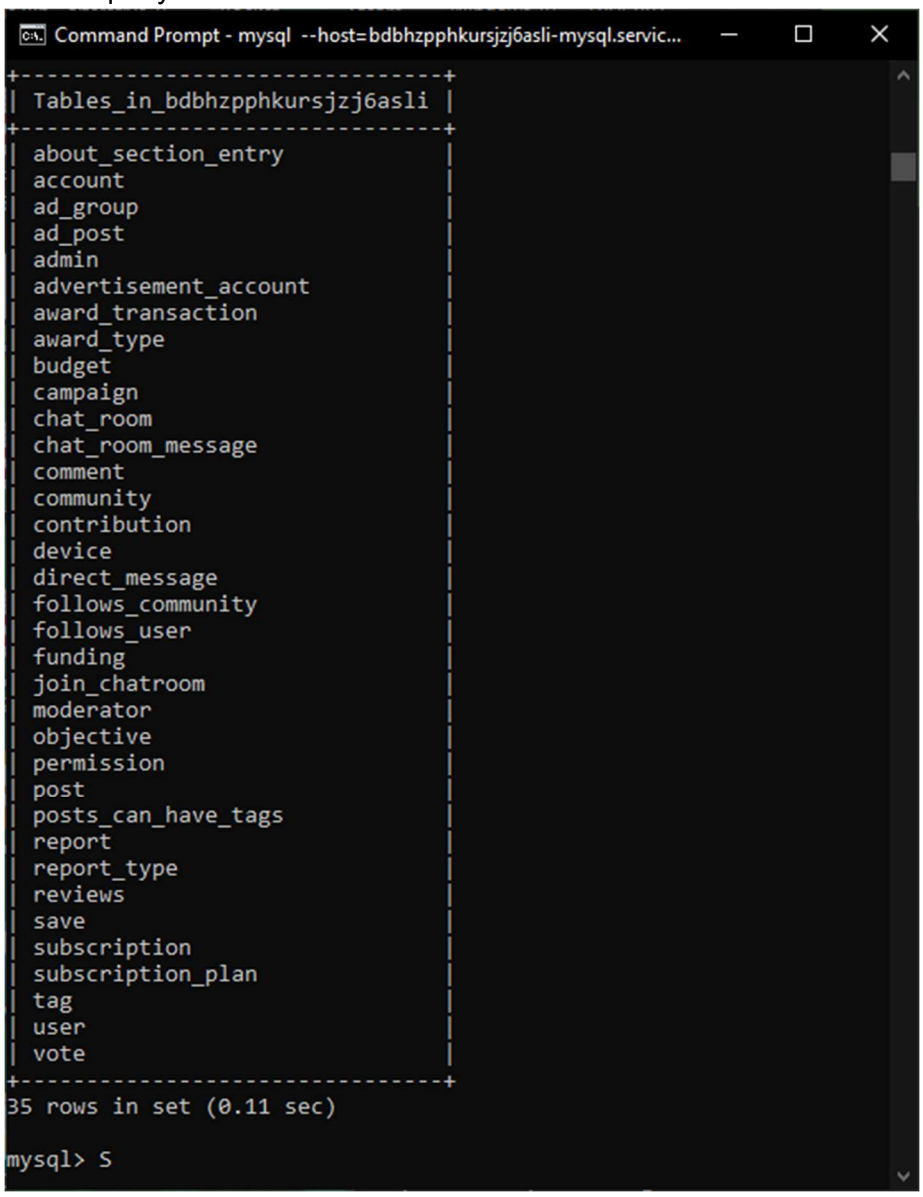
An important part to notice here is the constraint we added on line 15 in above snippet. For each user, the tuple (user_id, account_id) has to be unique. Like wise there were several instances where we needed this constraint.

We used command line interface and mysql server-client to interface with our database and execute our queries. Below we have attached few screenshots of us firing the create queries.





Finally, we have created all tables. Below we show the screenshot of cli showing the ‘show tables’ query.



Insertion of data.

Before we could start with insert queries, we need the most impotant thing-***data***! The challenge here was not to come up with mock data, but to create data for a database with 35 tables. So inorder to create mock data for our database and not have us waste time creating mock data, we made use of online available solutions such as **Mockaroo**. Using this, we were able to create high amounts of data. This tool also had us specify the datatypes and made the insertion queries for us.

Below we have a screenshot of insertion queries as an example

moderator.sql | follows_user.sql

```
insert into moderator (moderator_id, user_id, community_id) values (1, 19, 7);
insert into moderator (moderator_id, user_id, community_id) values (2, 17, 10);
insert into moderator (moderator_id, user_id, community_id) values (3, 20, 9);
insert into moderator (moderator_id, user_id, community_id) values (4, 11, 5);
insert into moderator (moderator_id, user_id, community_id) values (5, 3, 5);
insert into moderator (moderator_id, user_id, community_id) values (6, 4, 1);
insert into moderator (moderator_id, user_id, community_id) values (7, 14, 6);
insert into moderator (moderator_id, user_id, community_id) values (8, 7, 6);
insert into moderator (moderator_id, user_id, community_id) values (9, 19, 8);
insert into moderator (moderator_id, user_id, community_id) values (10, 14, 4);
insert into moderator (moderator_id, user_id, community_id) values (11, 5, 3);
insert into moderator (moderator_id, user_id, community_id) values (12, 14, 1);
insert into moderator (moderator_id, user_id, community_id) values (13, 2, 8);
insert into moderator (moderator_id, user_id, community_id) values (14, 14, 7);
insert into moderator (moderator_id, user_id, community_id) values (15, 19, 2);

```

Line 1, Column 1 | Tab Size: 4 | SQL

Again, we made use of mysql cli to fire our insertion queries. Below we have given few screenshots of our implementation of insert queries

Windows PowerShell

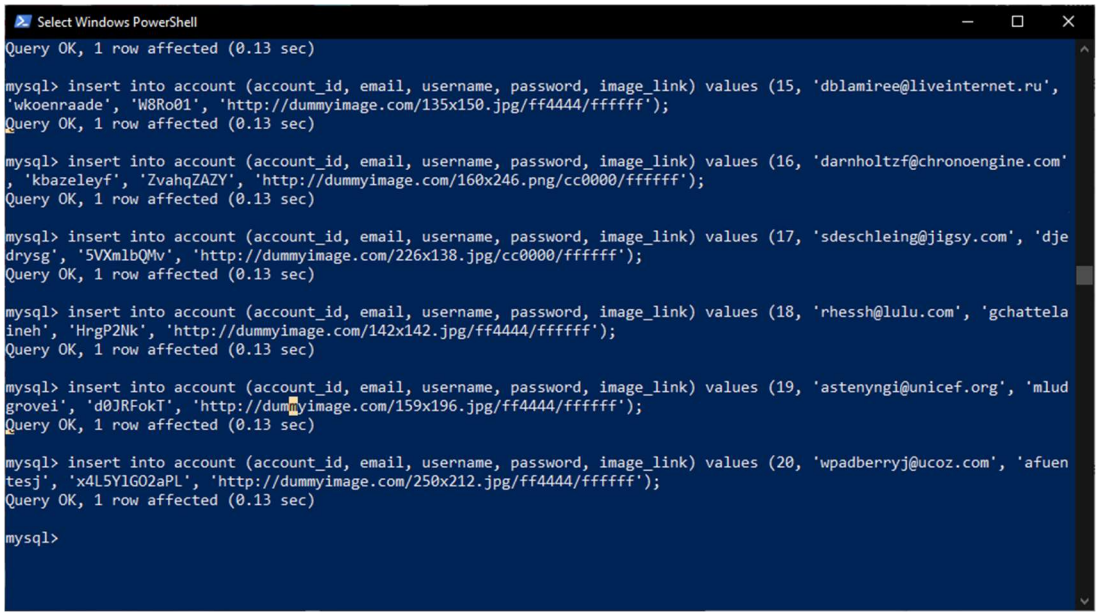
```
mysql> insert into campaign (campaign_id, name, funding_id, ad_account_id, objective_id) values (16, 'Y-find', 1, 8, 2);
Query OK, 1 row affected (0.13 sec)

mysql> insert into campaign (campaign_id, name, funding_id, ad_account_id, objective_id) values (17, 'Span', 1, 11, 1);
Query OK, 1 row affected (0.12 sec)

mysql> insert into campaign (campaign_id, name, funding_id, ad_account_id, objective_id) values (18, 'Wrapsafe', 2, 13, 2);
Query OK, 1 row affected (0.13 sec)

mysql> insert into campaign (campaign_id, name, funding_id, ad_account_id, objective_id) values (19, 'Mat Lam Tam', 2, 14, 4);
ERROR 1062 (23000): Duplicate entry '19' for key 'PRIMARY'
mysql> insert into campaign (campaign_id, name, funding_id, ad_account_id, objective_id) values (20, 'Rank', 1, 13, 1);
ERROR 1062 (23000): Duplicate entry '20' for key 'PRIMARY'
mysql> select * from campaign
    -> ;
+-------------+-------------+------------+---------------+-------------+
| campaign_id | name        | funding_id | ad_account_id | objective_id |
+-------------+-------------+------------+---------------+-------------+
|           1 | Job         |          2 |             3 |           4 |
|           2 | Temp        |          1 |             2 |           4 |
|           3 | Transcof    |          1 |             6 |           2 |
|           4 | Alpha       |          2 |             2 |           3 |
|           5 | Bamity      |          2 |             1 |           4 |
|           6 | Keylex      |          1 |            13 |           3 |
|           7 | Konklux     |          1 |            12 |           3 |
|           8 | Subin       |          2 |            18 |           3 |
|           9 | Voltsillam  |          1 |             6 |           5 |
|          10 | Alphazap    |          1 |            18 |           3 |
|          11 | Veribet     |          2 |            14 |           3 |
|          12 | Subin       |          2 |             1 |           1 |
|          13 | Quo Lux     |          1 |            19 |           3 |
|          14 | Flowdesk    |          1 |             9 |           2 |
|          15 | Regrant     |          1 |             7 |           1 |
|          16 | Y-find      |          1 |             8 |           2 |
|          17 | Span        |          1 |            11 |           1 |
|          18 | Wrapsafe    |          2 |            13 |           2 |
|          19 | Mat Lam Tam |          2 |            14 |           4 |
|          20 | Rank        |          1 |            13 |           1 |
+-------------+-------------+------------+---------------+-------------+
20 rows in set (0.11 sec)

mysql>
```



Now, with data inserted, we have **our database implementation completed**!

Problems faced during implementation

- We had initially hoped to use a cloud-based free mysql server, but unfortunately we had to switch to our local instance instead, due to restrictions on stored procedures, functions and triggers in free server providers
- Initially, we had not added any constraints, especially unique. This proved to really bad in later staged when we found the inserted data wasn't the way it was supposed to be. Issues we face were duplicates, ambiguous entries, null entries; in short the data made no sense. Since the we we found out about this after inserting the data, we couldn't alter the tables then; and we had to ultimately start over again.
- We initially started creating mock data on sql but it was too time consuming, and we would have never completed it on time. Hence we had to switch to mockaroo to create our data.

Queries

We have created queries for few specific use-cases such that they match some business rules. For instance, we have picked frequently utilized functions, in terms of data, of reddit like fetch latest trending posts, best comments and replies on post, all tags attached to each posts, etc.

We have structured our queries in decending order of complexity. These are as follows:

1. Some query
2. Another Query
3. LOL query
4. JNAIDNAODN
5. ADsoladoanad
6. Ds
7. Sad
8. As
9. Das
10. D
11. Asd
12. As
13. Das
14. Da
15. Sd
16. Asd
17. A
18. Ds
19. Ads
20. Asd
21. Asd
22. As
23. D
24. Sad
25. Adas

For each of these use cases, we have given below the query along with the screenshots of results after executing the query.

Execution of queries

1. The name of users which follow community Music.

select u.display_name from follows_community f,community c,user u where u.user_id=f.user_id and c.community_id=f.community_id and c.name='Music';

```
mysql> select u.display_name from follows_community f,community c,user u where u.user_id=f.user_id and c.community_id=f.
community_id and c.name='Music';
+--------------+
| display_name |
+--------------+
| Pris         |
| Gus          |
| Darlleen     |
| Laughton     |
| Libbi        |
| Tish         |
| Meg          |
+--------------+
7 rows in set (0.20 sec)
```

2. Find out such cost_type and campaign name that has objectives as ‘App Install’.

select name,cost_type from campaign c , objective o where o.objective_id=c.objective_id and o.objective_type='App Install';

```
mysql> select name,cost_type from campaign c , objective o where o.objective_id=c.objective_id and o.objective_type='App Install';
    -> \g
+------------+-----------+
| name       | cost_type |
+------------+-----------+
| Voltsillam | CPC       |
+------------+-----------+
1 row in set (0.00 sec)
```

3. Find out such adgroup who’s name doesn’t starts with “A”

select * from ad_group where not name like 'A%';

```
mysql> select * from ad_group where not name like 'A%';
+-------------+-------------+-----------+------------+-----------+-----------+-------------+
| ad_group_id | name        | location  | schedule   | budget_id | device_id | campaign_id |
+-------------+-------------+-----------+------------+-----------+-----------+-------------+
|           1 | Vagram      | France    | 2020-02-19 |         2 |         1 |           1 |
|           2 | Cardify     | Italy     | 2020-01-15 |         2 |         1 |           2 |
|           3 | Konklux     | China     | 2019-12-10 |         1 |         5 |           3 |
|           4 | Zathin      | Comoros   | 2020-05-31 |         1 |         1 |           4 |
|           5 | Toughjoyfax | Indonesia | 2020-02-03 |         1 |         2 |           5 |
|           6 | Rank        | China     | 2019-12-14 |         1 |         2 |           6 |
|           8 | Cardify     | Indonesia | 2020-05-03 |         2 |         2 |           8 |
|           9 | Sub-Ex      | Russia    | 2020-02-04 |         2 |         4 |           9 |
|          10 | Span        | China     | 2020-02-27 |         1 |         1 |          10 |
|          11 | Hatity      | Indonesia | 2020-03-04 |         2 |         5 |          11 |
|          12 | Home Ing    | Nigeria   | 2020-10-12 |         1 |         2 |          12 |
|          13 | Treeflex    | Indonesia | 2020-07-13 |         2 |         3 |          13 |
|          14 | Konklux     | Indonesia | 2020-11-05 |         2 |         2 |          14 |
|          15 | Solarbreeze | Finland   | 2020-11-10 |         1 |         3 |          15 |
|          16 | Stim        | Nigeria   | 2020-11-30 |         1 |         4 |          16 |
|          17 | Fixflex     | Greece    | 2020-11-07 |         1 |         5 |          17 |
|          18 | Holdlamis   | Morocco   | 2020-10-13 |         1 |         3 |          18 |
|          19 | Keylex      | Indonesia | 2019-12-23 |         1 |         4 |          19 |
|          20 | Wrapsafe    | Thailand  | 2020-12-03 |         2 |         5 |          20 |
+-------------+-------------+-----------+------------+-----------+-----------+-------------+
19 rows in set (0.19 sec)
```

4. Find out all campiagns which have credit card as funding type?

select name from campaign c,funding f where f.funding_id=c.funding_id and funding_type='credit card';

```
mysql> select name from campaign c,funding f where f.funding_id=c.funding_id and funding_type='credit card';
+--------------------------+
| name                     |
+--------------------------+
| Jerry Springer: Ringmaster |
| Temp                     |
| Transcof                 |
| Keylex                   |
| Konklux                  |
| Voltsillam               |
| Alphazap                 |
| Quo Lux                  |
| Flowdesk                 |
| Regrant                  |
| Y-find                   |
| Span                     |
| Rank                     |
+--------------------------+
13 rows in set (0.19 sec)
```

5. Find name of such ad which is located in pune and has budget as “lifetime”.

select a.name from ad_post a, budget b,ad_group d where a.ad_group_id=d.ad_group_id and d.budget_id=b.budget_id and d.location like 'Indonesia' and b.budget_type like 'Lifetime Budget';

```
mysql> select a.name from ad_post a, budget b,ad_group d where a.ad_group_id=d.ad_group_id and d.budget_id=b.budget_id a
nd d.location like 'Indonesia' and b.budget_type like 'Lifetime Budget';
+----------+
| name     |
+----------+
| Tres-Zap |
| Duobam   |
| Tres-Zap |
| Tres-Zap |
| Subin    |
+----------+
5 rows in set (0.19 sec)

mysql>
```

6. Find out such users where their about information is missing.

select * from user where about=null;

```
mysql> select * from user where about=null;
Empty set (0.19 sec)

mysql>
```

7. How many campaigns from CompanyName ‘IXEMPRA’ have objectives?

select count(c.campaign_id) from campaign c , advertisement_account a,objective o where o.objective_id=c.objective_id and c.ad_account_id=a.ad_account_id and a.Company_name='IXEMPRA';

```
mysql> select count(c.campaign_id) from campaign c , advertisement_account a,objective o where o.objective_id=c.objectiv
e_id and c.ad_account_id=a.ad_account_id and a.Company_name='IXEMPRA';
+----------------------+
| count(c.campaign_id) |
+----------------------+
|                    1 |
+----------------------+
1 row in set (0.20 sec)

mysql>
```

8. What was the last day of the month when Pris created the community?

select last_day(creation_date) from community c , user u where u.user_id=c.user_id and u.display_name='Pris';

```
mysql> select last_day(creation_date) from community c , user u where u.user_id=c.user_id and u.display_name='Pris';
+-------------------------+
| last_day(creation_date) |
+-------------------------+
| 2020-07-31              |
+-------------------------+
1 row in set (0.21 sec)

mysql>
```

9. Display Advertisement accounts in a ascending order of their company name.

select Company_name,industry from advertisement_account order by Company_name;

```
mysql> select Company_name,industry from advertisement_account order by Company_name;
+----------------------+---------------------------+
| Company_name         | industry                  |
+----------------------+---------------------------+
| Allopurinol          | Savings Institutions      |
| Allopurinol          | Savings Institutions      |
| Allopurinol          | Savings Institutions      |
| Enalapril Maleate    | Multi-Sector Companies    |
| Entacapone           | Major Pharmaceuticals     |
| health mart ibuprofen | Oil & Gas Production     |
| health mart ibuprofen | Oil & Gas Production     |
| health mart ibuprofen | Oil & Gas Production     |
| Idamycin PFS         | Coal Mining               |
| IXEMPRA              | Major Banks               |
| Lucky Super Soft     | Major Banks               |
| Midodrine HCl        | Business Services         |
| Muscle Ease          | Finance: Consumer Services |
| Shark Cartilage      | Business Services         |
| Shark Cartilage      | Business Services         |
| Shark Cartilage      | Business Services         |
| Value Pharma         | Electrical Products       |
| Vicks VapoRub        | Major Chemicals           |
| Vicks VapoRub        | Major Chemicals           |
| Vicks VapoRub        | Major Chemicals           |
+----------------------+---------------------------+
20 rows in set (0.19 sec)
```

10. Give device wise ad_group details.

select d.device_type, count(a.ad_group_id) from device d, ad_group a where a.device_id=d.device_id group by device_type;

```
mysql> select d.device_type, count(a.ad_group_id) from device d, ad_group a where a.device_id=d.device_id group by devic
e_type;
+-------------+----------------------+
| device_type | count(a.ad_group_id) |
+-------------+----------------------+
| Android     |                    5 |
| iOS         |                    5 |
| Windows     |                    3 |
| Mac         |                    3 |
| All         |                    4 |
+-------------+----------------------+
5 rows in set (0.19 sec)
```

11. Give NSFWFlag wise user details.

select u.nsfw_flag,count(p.user_id) from user u where group by u.nsfw_flag;

```
mysql> select u.nsfw_flag,count(u.user_id) from user u group by u.nsfw_flag;
    -> \g
+-----------+------------------+
| nsfw_flag | count(u.user_id) |
+-----------+------------------+
|         0 |                8 |
|         1 |               12 |
+-----------+------------------+
2 rows in set (0.20 sec)

mysql>
```

12. Print the about of a community whose name starts with B in capital.

select upper(a.title) from community c,about_section_entry a where a.community_id=c.community_id and c.name like 'B%';

```
mysql> select upper(a.title),c.name from community c,about_section_entry a where a.community_id=c.community_id and c.nam
e like 'M%';
+----------------+--------------+
| upper(a.title) | name         |
+----------------+--------------+
| HOLDLAMIS      | Music        |
| OVERHOLD       | Movies and Tv |
| FLOWDESK       | Memes        |
+----------------+--------------+
3 rows in set (0.19 sec)

mysql>
```

13. Find all the messages from table direct_message containing the word "Elementum"

select * from direct_messages where message like "%elementum%";

Functions

1. Write a SQL function to find total no. of ad accounts having permission level as creator.

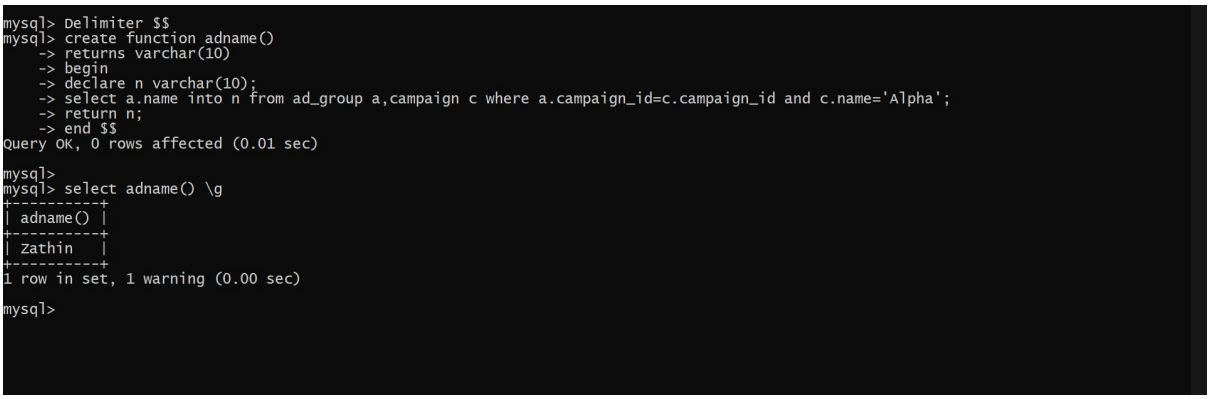
```
create function totalacc()
returns int
begin
declare n int;
select count(a.ad_account_id) into n from advertisement_account a,permission p where
p.permission_id=a.permission_id and p.permission_level='creator';
return n;
end $$

select totalacc() \g
```

2. Write a PL/SQL function to find the name of the all ad_groups based having alpha campaign

```
Delimiter $$
create function adname()
returns varchar(10)
begin
declare n varchar(10);
select a.name into n from ad_group a,campaign c where a.campaign_id=c.campaign_id and
c.name='Alpha';
return n;
end $$

select adname() \g
```


mysql> Delimiter $$
mysql> create function adname()
-> returns varchar(10)
-> begin
-> declare n varchar(10);
-> select a.name into n from ad_group a,campaign c where a.campaign_id=c.campaign_id and c.name='Alpha';
-> return n;
-> end $$
Query OK, 0 rows affected (0.01 sec)
mysql>
mysql> select adname() \g
+----------+
| adname() |
+----------+
| Zathin |
+----------+
1 row in set, 1 warning (0.00 sec)
mysql>

Procedure

1. Write a procedure to find details of a given community;

```
delimiter $$
create procedure commdet(in cname varchar(30))
begin
select * from about_section_entry a where a.community_id in (select community_id from community where name=cname);
end $$

call commdet('Fashion')\g
```

```
mysql> delimiter $$
mysql>  create procedure commdet(in cname varchar(30))
    ->  begin
    -> select * from about_section_entry a where a.community_id in (select community_id from community where name=cname)
;
    -> end $$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> call commdet('Fashion');
    -> \g
+----------+------------+----------+-----------------------------------------+-----------------------------------
---------------------------+--------------+
| about_id | date_added | title    | description                             | links
                           | community_id |
+----------+------------+----------+-----------------------------------------+-----------------------------------
---------------------------+--------------+
|       17 | 2020-05-26 | Overhold | disintermediate e-business infrastructures | https://robohash.org/ipsamaioresvolupt
as.jpg?size=50x50&set=set1 |            1 |
+----------+------------+----------+-----------------------------------------+-----------------------------------
---------------------------+--------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql>
```

```
mysql> delimiter $$
mysql> create procedure commdet(in cname varchar(30))
    -> begin
    -> select * from about_section_entry a where a.community_id in (select community_id from community where name=cname)
;
    -> end $$
Query OK, 0 rows affected (0.28 sec)

mysql> call commdet('Fashion');
    -> \g
+----------+------------+----------+-----------------------------------------+-----------------------------------
---------------------------+--------------+
| about_id | date_added | title    | description                             | links
                           | community_id |
+----------+------------+----------+-----------------------------------------+-----------------------------------
---------------------------+--------------+
|       17 | 2020-05-26 | Overhold | disintermediate e-business infrastructures | https://robohash.org/ipsamaioresvolupt
as.jpg?size=50x50&set=set1 |            1 |
+----------+------------+----------+-----------------------------------------+-----------------------------------
---------------------------+--------------+
1 row in set (0.20 sec)

Query OK, 0 rows affected, 1 warning (0.22 sec)

mysql>
```

2. Write a procedure to provide the name of user who is moderator of a particular community.

```
delimiter $$
create procedure nameus(in cname varchar(30))
begin
select display_name from moderator m,user u,community c where
m.community_id=c.community_id and u.user_id=m.user_id and c.name=cname;
end $$

call nameus('Cars');
```

```
mysql> delimiter $$
mysql> create procedure nameus(in cname varchar(30))
    -> begin
    -> select display_name from moderator m,user u,community c where m.community_id=c.community_id and u.user_id=m.user_id and c.name=c
name;
    -> end $$
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> call nameus('Cars')\g
+--------------+
| display_name |
+--------------+
| Leilah       |
+--------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected, 1 warning (0.03 sec)
```

3. Write a procedure to find out total no. of posts written by a particular date.

```
delimiter $$
create procedure postcount(in datename date)
begin
select count(p.post_id) from post p where contribution_id in (select c.contribution_id from
contribution c where c.date_posted=datename);
end $$

call postcount('2020-10-23')\g
```

function:

```
delimiter $$
create function postcon(datename date)
returns int
begin
declare n int;
select count(p.post_id) into n from post p where contribution_id in (select c.contribution_id from
contribution c where c.date_posted=datename);
```

return n;
end $$
call postcount('2020-10-23')\g

```
mysql> delimiter $$
mysql> create procedure postcount(in datename date)
    -> begin
    -> select count(p.post_id) from post p where contribution_id in (select c.contribution_id from contribution c where
c.date_posted=datename);
    -> end $$
Query OK, 0 rows affected (0.20 sec)

mysql> call postcount('2020-03-01 20:16:59');
    -> \g
+------------------+
| count(p.post_id) |
+------------------+
|                0 |
+------------------+
1 row in set (0.21 sec)

Query OK, 0 rows affected, 1 warning (0.21 sec)
```

```
mysql> call postcount('2020-10-23');
    -> \g
+------------------+
| count(p.post_id) |
+------------------+
|                1 |
+------------------+
1 row in set (0.20 sec)

Query OK, 0 rows affected (0.20 sec)
```

4. A procedure to edit text of an existing comment.

```
delimiter $$
CREATE PROCEDURE editComment (in con_id int, in text_to_be_added varchar(30))
BEGIN
UPDATE contribution
SET  text = concat(text,text_to_be_added)
WHERE contribution_id=con_id;
end $$
```

call editComment(2,' comment edited');

```
mysql> delimiter $$
mysql> CREATE PROCEDURE editComment (in con_id int, in text_to_be_added varchar(30))
    -> BEGIN
    -> UPDATE contribution
    -> SET  text = concat(text,text_to_be_added)
    -> WHERE contribution_id=con_id;
    -> end $$
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> SELECT *from contribution where contribution_id = '2';
+-----------------+-----------------------------------------------------------------------+-------------+---------+
| contribution_id | text                                                                  | date_posted | user_id |
+-----------------+-----------------------------------------------------------------------+-------------+---------+
|               2 | Integer ac leo. Pellentesque ultrices mattis odio. Donec vitae nisi. | 2020-07-20  |       3 |
+-----------------+-----------------------------------------------------------------------+-------------+---------+
1 row in set (0.01 sec)

mysql> call editComment(2,' comment edited');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SELECT *from contribution where contribution_id = '2';
+-----------------+-----------------------------------------------------------------------------------------+-------------+---------+
| contribution_id | text                                                                                    | date_posted | user_id |
+-----------------+-----------------------------------------------------------------------------------------+-------------+---------+
|               2 | Integer ac leo. Pellentesque ultrices mattis odio. Donec vitae nisi. comment edited | 2020-07-20  |       3 |
+-----------------+-----------------------------------------------------------------------------------------+-------------+---------+
1 row in set (0.00 sec)
```

5. Procedure to fetch details of a community.

```
delimiter $$
CREATE PROCEDURE community_details (in com_id int, out about_sectionID int, out
follower_count int )
BEGIN
select about_id from about_section_entry AS about_sectionID WHERE community_id =
com_id;
select COUNT(com_id) AS follower_count from follows_community;
end $$

call community_details(3,@about_sectionID,@follower_count);
select @about_sectionID;
select @follower_count;
```

```
mysql> use reddit;
Database changed
mysql> delimiter $$
mysql> CREATE PROCEDURE community_details (in com_id int, out about_sectionID int, out follower_count int )
    -> BEGIN
    -> select about_id from about_section_entry AS about_sectionID WHERE community_id = com_id;
    -> select COUNT(com_id) AS follower_count from follows_community;
    -> end $$
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> use reddit;
Database changed
mysql> call community_details('3',@about_sectionID, @follower_count);
+----------+
| about_id |
+----------+
|        6 |
+----------+
1 row in set (0.02 sec)

+----------------+
| follower_count |
+----------------+
|             65 |
+----------------+
1 row in set (0.03 sec)

Query OK, 0 rows affected (0.03 sec)
```

6. Procedure for notification of a direct message.

```
delimiter $$
create procedure dm_notification(in user_id int, out msg varchar(30), out sender_name
varchar(30))
begin
select message from direct_message as msg where user_one = user_id AND max(date) limit
1;
set @sender = 0;
select user_two from direct_message as sender where user_one = user_id AND max(date)
limit 1;
select display_name from user as sender_name where user_id = sender ;
end $$
```

```
mysql> use reddit;
Database changed
mysql> delimiter $$
mysql> create procedure dm_notification(in user_id int, out msg varchar(30), out sender_name varchar(30))
    -> begin
    -> select message from direct_message as msg where user_one = user_id AND max(date) limit 1;
    -> set @sender = 0;
    -> select user_two from direct_message as sender where user_one = user_id AND max(date) limit 1;
    -> select display_name from user as sender_name where user_id = sender ;
    -> end $$
Query OK, 0 rows affected (0.03 sec)
```

Triggers

1. Implement After update trigger on the Device Table.

```
create table updatedevice (upid int auto_increment,device_id int,device_type varchar(30),action
varchar(25),changedon datetime,primary key(upid));

delimiter $$
create trigger updatedevicetrigger
```

after update on device
for each row
begin
insert into updatedevice set action='Update',device_id=old.device_id ,
device_type=old.device_type ,changedon=now();
end $$

update device set device_type='Linux' where device_id=5\g

select * from updatedevice;

select * from device\g

```
mysql> create table updatedevice (upid int auto_increment,device_id int,device_type varchar(30),action varchar(25),changedon datetime,p
rimary key(upid));
Query OK, 0 rows affected (0.06 sec)

mysql>
mysql> delimiter $$
mysql> create trigger updatedevicetrigger
    -> after update on device
    -> for each row
    -> begin
    -> insert into updatedevice set action='Update',device_id=old.device_id , device_type=old.device_type ,changedon=now();
    -> end $$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> update device set device_type='Linux' where device_id=5\g
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> select * from updatedevice;
    -> \g
+------+-----------+-------------+--------+---------------------+
| upid | device_id | device_type | action | changedon           |
+------+-----------+-------------+--------+---------------------+
|    1 |         5 | All         | Update | 2020-12-11 15:45:54 |
+------+-----------+-------------+--------+---------------------+
1 row in set (0.00 sec)

mysql> select * from device\g
+-----------+-------------+
| device_id | device_type |
+-----------+-------------+
|         1 | Android     |
|         2 | iOS         |
|         3 | Windows     |
|         4 | Mac         |
|         5 | Linux       |
+-----------+-------------+
```

2. Implement before insert trigger on the funding Table.

create table updatefund (upf int auto_increment,funding_id int,funding_type varchar(15),action varchar(25),changedon datetime,primary key(upf))\g

delimiter $$
create trigger insertfundingtrigger
before insert on funding
for each row
begin
insert into updatefund set action='Insert' ,
funding_id=new.funding_id,funding_type=new.funding_type,changedon=now();
end $$

insert into funding values(3,'Paypal')\g

select * from updatefund\g

select * from funding\g

```
mysql> create table updatefund (upf int auto_increment,funding_id int,funding_type varchar(15),action varchar(25),changedon datetime,pr
imary key(upf))\g
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> delimiter $$
mysql> create trigger insertfundingtrigger
    -> before insert on funding
    -> for each row
    -> begin
    -> insert into updatefund set action='Insert' , funding_id=new.funding_id,funding_type=new.funding_type,changedon=now();
    -> end $$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> insert into funding values(3,'Paypal')\g
Query OK, 1 row affected (0.01 sec)

mysql> select * from updatefund\g
+-----+------------+--------------+--------+---------------------+
| upf | funding_id | funding_type | action | changedon           |
+-----+------------+--------------+--------+---------------------+
|   1 |          3 | Paypal       | Insert | 2020-12-11 15:47:58 |
+-----+------------+--------------+--------+---------------------+
1 row in set (0.00 sec)

mysql> select * from funding\g
+------------+--------------+
| funding_id | funding_type |
+------------+--------------+
|          1 | Credit Card  |
|          2 | Debit Card   |
|          3 | Paypal       |
+------------+--------------+
3 rows in set (0.00 sec)

mysql>
```

3. Implement before update trigger on the About Table.

create table updateabout (upaid int auto_increment,about_id int,title VARCHAR(50),description VARCHAR(50),action varchar(25),changedon datetime,primary key(upaid))\g

delimiter $$
create trigger updateabouttrigger
before update on about_section_entry
for each row
begin
insert into updateabout set action='Update',about_id=new.about_id ,
title=new.title,description=new.description ,changedon=now();
end $$

update about_section_entry set title='Overpower' where about_id=15\g

select * from updateabout\g

select * from about_section_entry\g

```
mysql> create table updateabout (upaid int auto_increment,about_id int,title VARCHAR(50),description VARCHAR(50),action varchar(25),cha
ngedon datetime,primary key(upaid))\g
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> delimiter $$
mysql> create trigger updateabouttrigger
    -> before update on about_section_entry
    -> for each row
    -> begin
    -> insert into updateabout set action='Update',about_id=new.about_id , title=new.title,description=new.description ,changedon=now()
;
    -> end $$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> update about_section_entry set title='Overpower' where about_id=15\g
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> select * from updateabout\g
+-------+----------+-----------+--------------------------------+--------+---------------------+
| upaid | about_id | title     | description                    | action | changedon           |
+-------+----------+-----------+--------------------------------+--------+---------------------+
|     1 |       15 | Overpower | implement synergistic portals | Update | 2020-12-11 15:50:03 |
+-------+----------+-----------+--------------------------------+--------+---------------------+
1 row in set (0.00 sec)
```

```
mysql> select * from about_section_entry\g
+----------+------------+-------------+-----------------------------------------+-----------------------------------------------------
-----------------+--------------+
| about_id | date_added | title       | description                             | links
                 | community_id |
+----------+------------+-------------+-----------------------------------------+-----------------------------------------------------
-----------------+--------------+
|        1 | 2020-01-31 | Overhold    | enable holistic e-tailers               | https://robohash.org/voluptatemetnulla.jpg?size=50
x50&set=set1     |            2 |
|        4 | 2020-05-04 | Y-Solowarm  | matrix 24/365 infrastructures           | https://robohash.org/earumsitqui.png?size=50x50&se
t=set1           |            7 |
|        5 | 2020-10-24 | Flowdesk    | enhance 24/7 vortals                    | https://robohash.org/sitsapienterecusandae.jpg?siz
e=50x50&set=set1 |           10 |
|        6 | 2020-10-02 | Andalax     | leverage real-time mindshare            | https://robohash.org/autemofficiacorporis.bmp?size
=50x50&set=set1  |            3 |
|        7 | 2020-10-03 | Biodex      | utilize cross-media e-commerce          | https://robohash.org/explicabodolorplaceat.png?siz
e=50x50&set=set1 |            8 |
|        8 | 2020-09-28 | Holdlamis   | seize B2B ROI                           | https://robohash.org/voluptasquasearum.jpg?size=50
x50&set=set1     |            5 |
|        9 | 2020-02-07 | Toughjoyfax | monetize wireless systems               | https://robohash.org/illoaperiamquia.png?size=50x5
0&set=set1       |            4 |
|       10 | 2020-03-20 | Opela       | envisioneer B2B action-items            | https://robohash.org/officiissedillum.bmp?size=50x
50&set=set1      |            9 |
|       15 | 2020-03-04 | Overpower   | implement synergistic portals           | https://robohash.org/ullamvelomnis.png?size=50x50&
set=set1         |            6 |
|       17 | 2020-05-26 | Overhold    | disintermediate e-business infrastructures | https://robohash.org/ipsamaioresvoluptas.jpg?size=
50x50&set=set1   |            1 |
+----------+------------+-------------+-----------------------------------------+-----------------------------------------------------
-----------------+--------------+
10 rows in set (0.00 sec)

mysql>
```

THANK YOU

Safeer
Ritvik
Rahul
Priyanshu

functions:
1. calculate karma-- post=10, comment=5, vote=2, award=8 -DONE
2. to functions to distinguish btween comment or post. DONE
3. Write a function to find out total no. of posts written by a particular date. DONE
4. for a contribution, calculate net vote.(contribution_id){} DONE
5. return count of all ad posts in a particular country(IN location) DONE

procedures:
1. notification for contribution => (user ,OUT post_or_comment, user_id, contribution_id, text) DONE
2. Write a procedure to find details of a given community => (IN CommunityID, OUT aboutsection_id, count_of_posts, count_of_followers) DONE
3. notification for dm =>(IN userjiska hai account, OUT user kisne bheja, OUT message) Doing but you may also try
4. get community of the week =>(OUT community_id) {for every contribution on a duration of week, you calculate using net vote votes recied for the contributions belonging there.} FAILED
5.edit comment->(IN contribution_id, IN text_to_be_added){ you have just add the text to existing text.} DONE

Triggers:
1. AUTO REPORT REVIEW; DONE
2. points to in comment add trigger
3. Implement After update trigger on the Device Table. ALREADY DONE
4. trigger at create community, needs specif karma to create- DONE
5. cannot create post in same community again on same day! DONE
6. cannot create post with text if link is video or image.? DOING……NEED HELP!
7. Trigger for giveing subscription to users with awards recieved above 1000.
8. Trigger to user table for account to exist and not repeat. DONE

Queries
. get all users whose subcription is due to end within a 2weeks. DONE
. treanding posts DONE
. popular comments
. fetch controversal posts/comments. DONE
. Find all the users who have upvoted a particular advertisement/company/Industry ~~DOING~~ NOT POSSIBLE
. All ad posts grouped by regions- useful for analytics
.