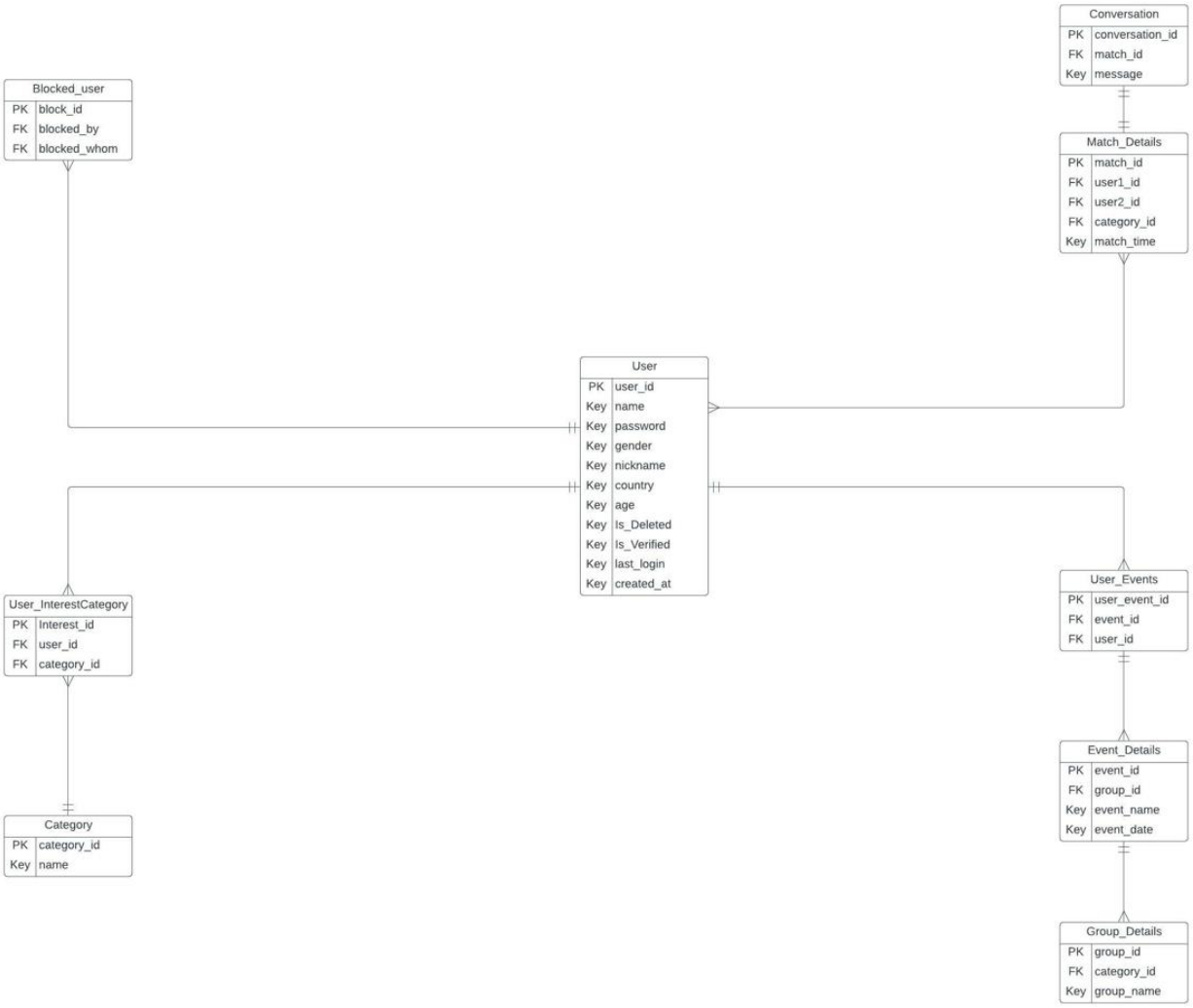
**Assignment 4 - Normalization (Sociable)**

## Team members:


1. Harshila Jagtap
   NUID : 002743674
2. Yash Pawar
   NUID : 002747371


# About

Today we are in the 21st century, where we can see a steep rise in people migrating to different places for study, work, lifestyle, and travel. They find it difficult to socialize and make new friends. Moreover, it is a difficult and cumbersome experience to find people and groups with similar interests in a short span of time. Socialize presents its users to easily find like-minded people amongst unknowns, quickly socialize and join social groups, attend events, converse, etc. This project aims to gather data, analyze and recommend like-minded suggestions by analyzing people's interests, and present statistics on the same.




**List of Tables :**

1. User
2. Blocked_User
3. User_Interest_Category
4. Category
5. Conversation
6. Match_Details
7. User_Events
8. Event_Details
9. Group_Details

- Normalization is the process of organizing the data in the database.

**Why do we need Normalization?**

- The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

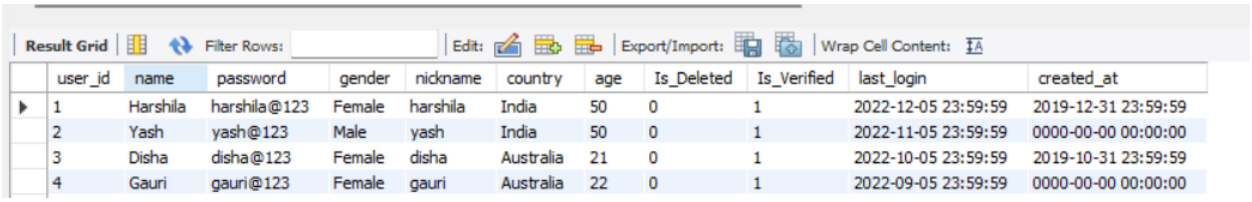
## First normal form (1NF)

- Each table has a primary key: minimal set of attributes which can uniquely identify a record

```
CREATE TABLE User (
user_id int not null AUTO_INCREMENT,
name varchar(255),
password varchar(255),
gender varchar(255),
nickname varchar(255),
country varchar(255),
age int,
Is_Deleted boolean,
Is_Verified boolean,
last_login timestamp,
created_at timestamp,
PRIMARY KEY (user_id)
);
```

➔ Each table above has a primary key displayed with PK. It will uniquely identify a record.

- The values in each column of a table are atomic (No multi-value attributes allowed).



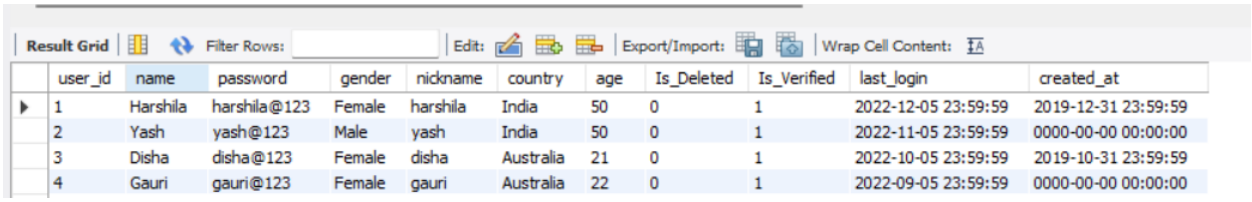
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| user_id | name | password | gender | nickname | country | age | Is_Deleted | Is_Verified | last_login | created_at |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Harshila | harshila@123 | Female | harshila | India | 50 | 0 | 1 | 2022-12-05 23:59:59 | 2019-12-31 23:59:59 |
| 2 | Yash | yash@123 | Male | yash | India | 50 | 0 | 1 | 2022-11-05 23:59:59 | 0000-00-00 00:00:00 |
| 3 | Disha | disha@123 | Female | disha | Australia | 21 | 0 | 1 | 2022-10-05 23:59:59 | 2019-10-31 23:59:59 |
| 4 | Gauri | gauri@123 | Female | gauri | Australia | 22 | 0 | 1 | 2022-09-05 23:59:59 | 0000-00-00 00:00:00 |

➔ The values that are stored in

- There are no repeating groups: two columns do not store similar information in the same table.

➔ No two columns store similar information



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

| user_id | name | password | gender | nickname | country | age | Is_Deleted | Is_Verified | last_login | created_at |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Harshila | harshila@123 | Female | harshila | India | 50 | 0 | 1 | 2022-12-05 23:59:59 | 2019-12-31 23:59:59 |
| 2 | Yash | yash@123 | Male | yash | India | 50 | 0 | 1 | 2022-11-05 23:59:59 | 0000-00-00 00:00:00 |
| 3 | Disha | disha@123 | Female | disha | Australia | 21 | 0 | 1 | 2022-10-05 23:59:59 | 2019-10-31 23:59:59 |
| 4 | Gauri | gauri@123 | Female | gauri | Australia | 22 | 0 | 1 | 2022-09-05 23:59:59 | 0000-00-00 00:00:00 |

**Hence, eliminated repeating groups as per 1NF.** Also the relation is in 1NF since it contains an atomic value.

---

A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.

## Second normal form (2NF)

- All requirements for 1st NF must be met.

➔ Done as per above screenshots of sample table

- No partial dependencies.

➔ There exists no partial dependencies

- No calculated data

➔ There exists no calculated data

---

### 3NF

A relation will be in 3NF if it is in 2NF and no transition dependency exists.

- All requirements for 2nd NF must be met.

  →All requirements met

- Eliminate fields that do not directly depend on the primary key; that is no transitive dependencies.
  → As per above design, no fields directly depend on the primary key. Hence there is no transitive dependency.

  As per design accordingly to meet 3NF, each table as shown above has Primary key. Also, necessary keys referred in other tables are marked as FK (foreign keys). No table can be broken further. Each table has its own purpose.

  Hence, our schema meets all the 3NF requirements.

# Views for all the use cases mentioned in Assignment 3

1. Use Case: View Groups with specific interest category

Description: User views a list of groups with mentioned interest

Actors: User

Precondition: User must specify his interests and there must be groups present

Steps:

Actor action – User views groups with specified interests

System Responses – groups would be displayed

Post Condition: system displays group list

View :

```
CREATE VIEW  View_Groups AS
SELECT * from Group_Details
INNER JOIN Category
ON group_Details.category_id=Category.category_id
Where Category.name LIKE "Basketball";
```

---

2. Use Case: View Events attended by specific user

Description: User views a list of events attended

Actors: User

Precondition: User must attend an event

Steps:

Actor action – User views events attended

System Responses – events would be displayed

Post Condition: system displays attended event list

View :

```
CREATE VIEW  View_Events AS
SELECT ed.event_name,ed.event_date from Event_Details ed
```

INNER JOIN User_Events ue

ON ed.event_id=ue.event_id

WHERE ue.user_id=2;

---

3. Use Case: View Matched User Details

Description: User views a list of users matched

Actors: User

Precondition: User must match another user

Steps:

Actor action – User views matched users

System Responses –Matched User List would be displayed

Post Condition: system displays Matched User list

View :

CREATE VIEW Matched_User AS

SELECT u.nickname,u.country,u.age from User u

INNER JOIN Match_Details md

ON u.user_id=md.user1_id

where u.user_id=2;

---

4.Use Case: View Users with specific interest

Description: User views a list of users with mentioned interest

Actors: User

Precondition: User must have mentioned interests

Steps:

Actor action – User views other users with BasketBall interest

System Responses –Matched User List with specific list would be displayed

Post Condition: system displays Matched User list

View :

```
CREATE VIEW  Specific_Interest AS
SELECT u.name,u.nickname,u.country,u.age from User u
INNER JOIN User_InterestCategory ui
ON u.user_id=ui.user_id
INNER JOIN Category c
ON ui.category_id=c.category_id
where c.name LIKE "Basketball" AND u.user_id !=2;
```

---

5.Use Case: View list of Blocked users for specificed user

Description: User views a list of blocked users

Actors: User

Precondition: User should block atleast 1 user

Steps:

Actor action – User views other blocked users

System Responses –Matched User List with specific list would be displayed

Post Condition: system displays Matched User list

View :

```
CREATE VIEW  Blocked_Users AS
SELECT u.name,u.nickname,u.country,u.age from User u
INNER JOIN Blocked_user bu
ON u.user_id=bu.blocked_whom
where bu.blocked_by=2;
```

---

6.Use Case: View list of Matched users in the month of September

Description: admin views a list of matched users in september

Actors: admin

Precondition: there should be matches in database

Steps:

Actor action – admin views list of Matched users in the month of September

System Responses –Matched User List with specific month would be displayed

Post Condition: system displays Matched User list for September

View :

```
CREATE VIEW  Matched_Users AS
SELECT md.user1_id,md.user2_id from Match_Details md
INNER JOIN User u
ON u.user_id=md.user1_id OR u.user_id=md.user2_id
where MONTH(md.match_time)=09;
```

---

7.Use Case: View list of Matched users with interest category Basketball

Description: admin views a list of matched users with common interest Category

Actors: admin

Precondition: there should be matches in database

Steps:

Actor action – admin views list of Matched users with common interest

System Responses –Matched User List with common interest would be displayed

Post Condition: system displays Matched User list

View :

```
CREATE VIEW  Specific_Interest_Category AS
SELECT md.user1_id,md.user2_id from Match_Details md
INNER JOIN User u
ON u.user_id=md.user1_id OR u.user_id=md.user2_id
INNER JOIN Category c
ON md.category_id=c.category_id
where c.name LIKE "Basketball";
```

---

8.Use Case: View Conversation of Matched users along with their ids

Description: admin views Conversation of matched users

Actors: admin

Precondition: there should be matches in database

Steps:

Actor action – admin views conversation of Matched users with common interest

System Responses –Matched User List along with their messages would be displayed

Post Condition: system displays Matched User list with text messages

View :

```
CREATE VIEW  Matched_User_Conversation AS
SELECT md.user1_id,md.user2_id, c.message from Match_Details md
INNER JOIN Conversation c
ON c.match_id=md.match_id;
```

---

9.Use Case: View groups with similar interest categories of a user

Description: User views List of groups with similar interest category

Actors: User

Precondition: there should be groups in database

Steps:

Actor action – User views List of groups with similar interest category

System Responses –Group List having having same interests

Post Condition: system displays list of groups

View :

```
CREATE VIEW  User_Similar_Interest AS
```

SELECT g.group_name from User u

INNER JOIN User_InterestCategory ui

ON u.user_id=ui.user_id

INNER JOIN Group_Details g

ON g.category_id=ui.category_id

where u.user_id=2;

---

10. Use Case: View events which share common liking with given user

Description: User views List of events with common interest category Actors: User

Precondition: there should be events in database

Steps:

Actor action – User views List of events with similar interest category

System Responses –event List having having same interests

Post Condition: system displays list of events

View :

CREATE VIEW Common_Category_Interest AS

SELECT g.group_name, ed.event_name, ed.event_date from User u

INNER JOIN User_InterestCategory ui

ON u.user_id=ui.user_id

INNER JOIN Group_Details g

ON g.category_id=ui.category_id

INNER JOIN Event_Details ed

ON g.group_id=ed.group_id

where u.user_id=2;