PERIOD TRACKER STUDY REPORT

Group No: 8

Student Names: Hanisha Reddy Cattamanchi Gopinath & Sakshi Pawar

Executive Summary:

The ***Personalized Period Tracker with Analytics and Social Sync*** is a comprehensive relational database solution designed to address the growing need for structured menstrual health tracking, predictive analytics, and socially aware cycle insights. Menstrual health applications typically operate on surface-level logging without leveraging deeper analytical modeling.

Many period-tracking apps only collect dates, leaving people without real clarity about what their bodies are doing. This project aims to close that gap by building a reliable, well-structured database that actually captures the full picture: cycles, symptoms, flow levels, medications, reminders, and even how users' cycles line up with their friends. Instead of just storing information, the system is designed to make that data useful.

At its heart, the system is meant to make menstrual tracking feel clearer and more supportive. Instead of leaving users to interpret scattered entries on their own, it pulls everything together into simple patterns and insights. This makes it easier to spot when something feels off, understand how cycles change over time, and trust the predictions they're getting.

The part that really makes this system stand out is the friendship-sync feature. Periods are personal, but they're also something a lot of friends end up dealing with around the same time anyway. Being able to share or compare timelines with someone you trust adds a small layer of comfort, whether you're trying to plan things better or just want a friend to check in on you when you're not feeling great. It makes the whole experience feel a bit more connected and supportive, instead of something you're expected to figure out by yourself.

Using an EER - driven conceptual foundation, the project incorporates multiple interacting components such as period records, cycles, symptom logs, flow logs, analytics specializations, and relationship-driven synchronization. These were systematically mapped into a fully normalized relational schema and implemented using MySQL. Realistic random data was generated to simulate real-world use cases, supporting both analytic queries and visualization workflows.

The database supports advanced analytical capabilities, including cycle regularity scoring, symptom frequency detection, flow - symptom correlations, prediction windows, and friend-based overlap calculations. These analytics not only provide actionable insights to individual users but also demonstrate the feasibility of scaling this architecture for larger menstrual health applications or integrating clinical decision-support systems.

I. Introduction

Menstrual cycle tracking is an important part of personal health management, helping individuals monitor fertility, recognize hormonal shifts, and notice early signs of irregularities. Most existing apps, however, offer only basic features. They record dates and send routine reminders but rarely provide meaningful analysis or help users understand long-term patterns. Even though cycle synchronization among people who spend time together has been widely discussed in research, mainstream tools do not include any form of social or comparative tracking.

This project, the Personalized Period Tracker with Analytics and Social Sync, was developed to close those gaps. The system collects detailed information such as period dates, daily flow, symptoms with severity scores, and medication usage. With several months of historical data, it becomes possible to calculate cycle averages, detect irregularities, identify recurring symptom trends, and generate predictions that reflect each user's real patterns rather than generic assumptions.

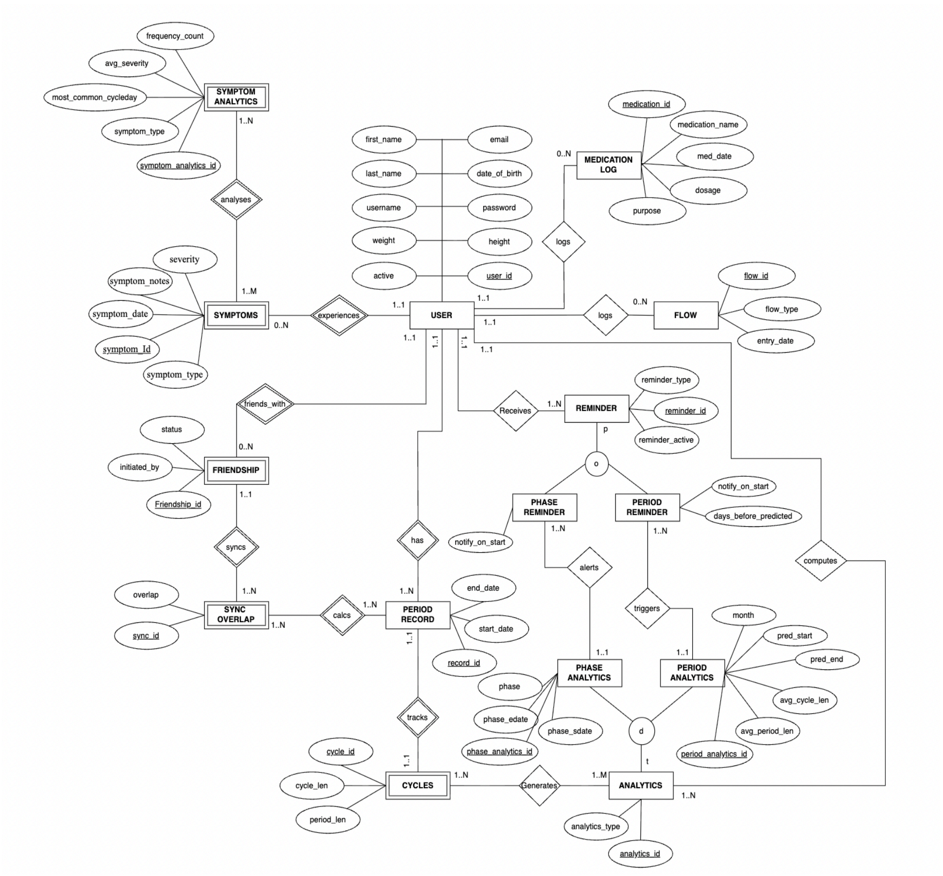
One feature that sets this system apart is its social sync functionality. Users can add friends, share limited cycle information, and view overlapping timelines. This creates opportunities for emotional support, easier planning, and a clearer understanding of shared or shifting symptoms.

The database supporting the platform was created through a structured design process. Requirements were translated into an EER model that captured key entities, relationships, constraints, and any necessary specializations. This conceptual model was then converted into a relational schema that follows normalization standards and enforces referential integrity. The final design was implemented in MySQL and tested with realistic synthetic data to validate both the structure and the analytical capabilities.

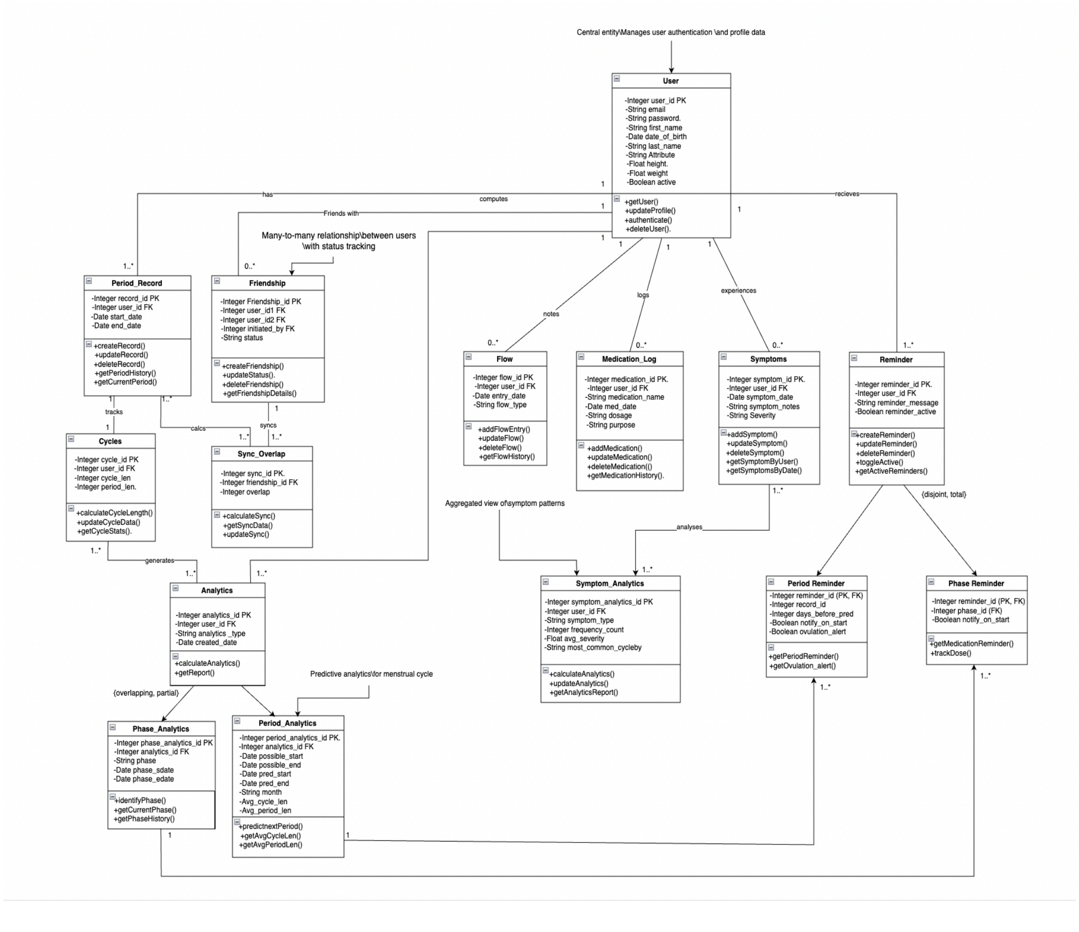
Overall, this project demonstrates how a carefully designed database can transform menstrual health information into practical insights. When data is captured consistently and analyzed with intention, it can reveal patterns that improve prediction accuracy, support better decision-making, and contribute to a more informed and comfortable experience for users.

II. Conceptual Data Modeling

1. EER Diagram



2. UML Diagram



III. Mapping Conceptual Model to Relational Model

Primary Key- Underlined **Foreign Key- *Italicized***

User (user_id, first_name, last_name, email, password, username, date_of_birth, weight, height, active)
- user_id is a primary key
- email and username have UNIQUE constraints
- All attributes except weight and height are NOT NULL

Friendship (friendship_id, *user_id_1, user_id_2*, status, *initiated_by*)
- Primary key : (user_id, friendship_id) ← Composite key
- user_id_1 is a foreign key referencing User (user_id) with NOT NULL values
- user_id_2 is a foreign key referencing User (user_id) with NOT NULL values
- initiated_by is a foreign key referencing User (user_id) with NOT NULL values
- status has CHECK constraint: ('Pending', 'Active', 'Blocked')

Sync_Overlap (sync_id, *friendship_id*, *user_id*, overlap)
- Primary key : (user_id, sync_id) ← Composite key
- friendship_id is a foreign key referencing Friendship (friendship_id) with NOT NULL values

Medication_Log (medication_id, *user_id*, medication_name, med_date, dosage, purpose)
- medication_id is a primary key
- user_id is a foreign key referencing User (user_id) with NOT NULL values
- dosage and purpose allow NULL values

Flow (flow_id, *user_id*, flow_type, entry_date)
- flow_id is a primary key
- user_id is a foreign key referencing User (user_id) with NOT NULL values
- flow_type has CHECK constraint: ('Light', 'Medium', 'Heavy', 'Spotting')

Symptoms (symptom_id, *user_id*, symptom_type, symptom_date, severity, symptom_notes)
- Primary Key: (user_id, symptom_id) ← Composite key
- user_id is a foreign key referencing User(user_id) with NOT NULL values
- symptom_type has CHECK constraint: ('Cramps', 'Headache', 'Bloating', 'Mood Swings', 'Fatigue', 'Acne', 'Back Pain', 'Nausea', 'Hot Flashes', 'Insomnia', 'Anxiety', 'Depression') with NOT NULL values
- severity has CHECK constraint: ('Mild', 'Moderate', 'Severe') with NOT NULL values
- symptom_notes allow NULL values

Symptom_Analytics (symptom_analytics_id, *user_id*, symptom_type, frequency_count, avg_severity, most_common_cycleday)
- Primary Key: (user_id, symptom_analytics_id) ← Composite key
- user_id is PART OF the primary key AND a foreign key referencing User (user_id) with NOT NULL values
- symptom_analytics_id is PART OF the primary key with NOT NULL values
- symptom_type has CHECK constraint: ('Cramps', 'Headache', 'Bloating', 'Mood Swings', 'Fatigue', 'Acne', 'Back Pain', 'Nausea', 'Hot Flashes', 'Insomnia', 'Anxiety', 'Depression') with NOT NULL values
- frequency_count with NOT NULL values (default: 0)
- avg_severity allows NULL values
- most_common_cycle allows NULL values

Period_Record (record_id, *user_id*, start_date, end_date)
- Primary Key: (user_id, record_id) ← Composite key
- user_id is PART OF the primary key AND a foreign key referencing User (user_id) with NOT NULL values
- record_id is PART OF the primary key with NOT NULL values
- start_date with NOT NULL values
- end_date allows NULL values
- CHECK constraint: end_date >= start_date (when end_date is not NULL)

Cycles (cycle_id, *record_id*, cycle_len, period_len)
- Primary Key: (record_id, cycle_id) ← Composite key
- record_id is a foreign key referencing Period_Record (record_id) with NOT NULL values
- cycle_id is PART OF the primary key with NOT NULL values
- cycle_len with NOT NULL values
- period_len with NOT NULL values
- CHECK constraint: cycle_len > 0 AND period_len > 0

Analytics (analytics_id, *cycle_id*, analytics_type)
- analytics_id is a primary key
- cycle_id is a foreign key referencing Cycles (cycle_id) with NOT NULL values
- analytics_type has CHECK constraint: ('Phase', 'Period')
- This is a superclass in a Total, Disjoint specialization

Phase_Analytics (phase_analytics_id, *analytics_id*, phase, phase_sdate, phase_edate)
- Primary Key: (phase_analytics_id)
- phase_analytics_id is a primary key with NOT NULL values
- analytics_id is a foreign key referencing Analytics(analytics_id) with NOT NULL values
- This table is a specialization (subclass) of Analytics, where each Phase_Analytics row corresponds to one Analytics row
- phase has a CHECK constraint: ('Menstrual', 'Follicular', 'Ovulation', 'Luteal') with NOT NULL values
- phase_sdate with NOT NULL values
- phase_edate allows NULL values
- CHECK constraint: phase_edate >= phase_sdate (when phase_edate is NOT NULL)

Period_Analytics (period_analytics_id, *analytics_id*, month, pred_start, pred_end, avg_cycle_len, avg_period_len)
- Primary Key: (period_analytics_id)
- period_analytics_id is a primary key with NOT NULL values
- analytics_id is a foreign key referencing Analytics(analytics_id) with NOT NULL values
- This table is a specialization (subclass) of Analytics, where each Period_Analytics row corresponds to one Analytics row
- month has a CHECK constraint: ('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December') with NOT NULL values
- pred_start allows NULL values
- pred_end allows NULL values
- CHECK constraint: pred_end >= pred_start (when both values are NOT NULL)
- avg_cycle_len with NOT NULL values
- avg_period_len with NOT NULL values
- CHECK constraint: avg_cycle_len > 0 AND avg_period_len > 0

Reminder (reminder_id, *user_id*, reminder_type, reminder_active)
- reminder_id is a primary key
- user_id is a foreign key referencing User (user_id) with NOT NULL values
- reminder_type has CHECK constraint: ('Phase', 'Period')
- This is a superclass in a Total, Disjoint specialization

Phase_Reminder (*reminder_id*, notify_on_start, phase)
- reminder_id is both a primary key AND a foreign key referencing Reminder(reminder_id) with NOT NULL values
- phase has CHECK constraint: ('Menstrual', 'Follicular', 'Ovulation', 'Luteal') with NOT NULL values
- This is a subclass of Reminder
- Period_Reminder (reminder_id, notify_on_start, days_before_predicted)
- reminder_id is both a primary key AND a foreign key referencing Reminder (reminder_id) with NOT NULL values
- This is a subclass of Reminder
- CHECK constraint: days_before_predicted >= 0

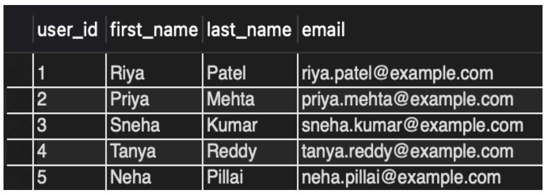
IV. Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation:

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

Query 1: Find users info who are stored in the database.

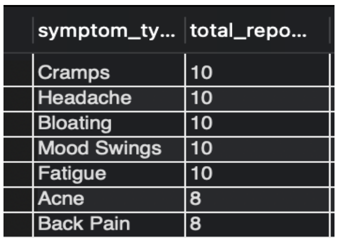
```
SELECT user_id, first_name, last_name, email
FROM User
LIMIT 10;
```



user_id	first_name	last_name	email
1	Riya	Patel	riya.patel@example.com
2	Priya	Mehta	priya.mehta@example.com
3	Sneha	Kumar	sneha.kumar@example.com
4	Tanya	Reddy	tanya.reddy@example.com
5	Neha	Pillai	neha.pillai@example.com

Query 2: Count symptom reports by type.

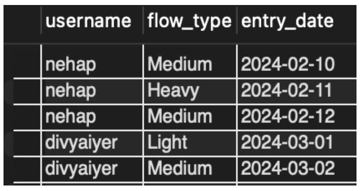
```
SELECT symptom_type, COUNT(*) AS total_reports
FROM Symptoms
GROUP BY symptom_type
ORDER BY total_reports DESC;
```



symptom_ty...	total_repo...
Cramps	10
Headache	10
Bloating	10
Mood Swings	10
Fatigue	10
Acne	8
Back Pain	8

Query 3: Find flow logs along with the username.

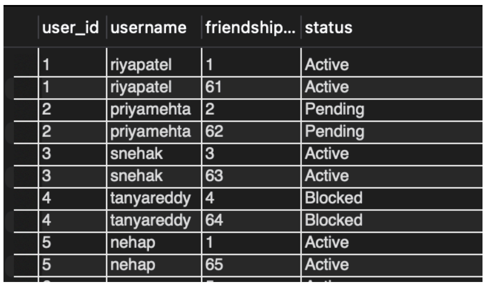
```
SELECT u.username, f.flow_type, f.entry_date
FROM Flow f
INNER JOIN User u ON f.user_id = u.user_id
```



username	flow_type	entry_date
nehap	Medium	2024-02-10
nehap	Heavy	2024-02-11
nehap	Medium	2024-02-12
divyaiyer	Light	2024-03-01
divyaiyer	Medium	2024-03-02

Query 4: Show all users along with any friendships (including users with no friendships).

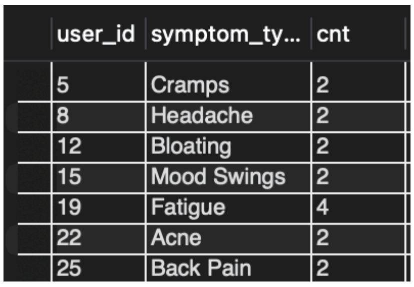
```
SELECT
    u.user_id,
    u.username,
    f.friendship_id,
    f.status
FROM User u
LEFT JOIN Friendship f
      ON (u.user_id = f.user_id_1 OR u.user_id = f.user_id_2)
ORDER BY u.user_id;
```



user_id	username	friendship...	status
1	riyapatel	1	Active
1	riyapatel	61	Active
2	priyamehta	2	Pending
2	priyamehta	62	Pending
3	snehak	3	Active
3	snehak	63	Active
4	tanyareddy	4	Blocked
4	tanyareddy	64	Blocked
5	nehap	1	Active
5	nehap	65	Active

Query 5: Find the most common symptom reported by each user.

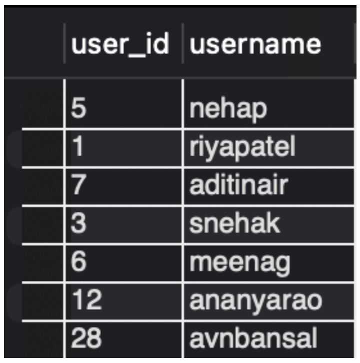
```
SELECT
user_id, symptom_type, COUNT(*) AS cnt
FROM Symptoms
GROUP BY user_id, symptom_type
HAVING COUNT(*) = (
    SELECT MAX(sub.cnt)
    FROM (
        SELECT user_id, symptom_type, COUNT(*) AS cnt
        FROM Symptoms
        GROUP BY user_id, symptom_type
    ) AS sub
    WHERE sub.user_id = Symptoms.user_id
);
```



user_id	symptom_ty...	cnt
5	Cramps	2
8	Headache	2
12	Bloating	2
15	Mood Swings	2
19	Fatigue	4
22	Acne	2
25	Back Pain	2

Query 6: Find users who share at least one active friend with another user.

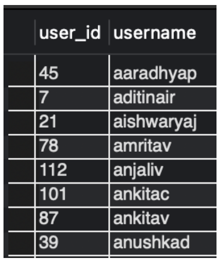
```
SELECT
u1.user_id,
u1.username
FROM User u1
WHERE EXISTS (
   SELECT 1
   FROM Friendship f1
   JOIN Friendship f2
       ON f1.user_id_1 = f2.user_id_1
       OR f1.user_id_1 = f2.user_id_2
       OR f1.user_id_2 = f2.user_id_1
       OR f1.user_id_2 = f2.user_id_2
   WHERE f1.status = 'Active'
     AND f2.status = 'Active'
     AND (f1.user_id_1 = u1.user_id OR f1.user_id_2 = u1.user_id)
     AND (f2.user_id_1 != u1.user_id AND f2.user_id_2 != u1.user_id)
);
```



user_id	username
5	nehap
1	riyapatel
7	aditinair
3	snehak
6	meenag
12	ananyarao
28	avnbansal

Query 7: Find users who have never logged heavy flow.

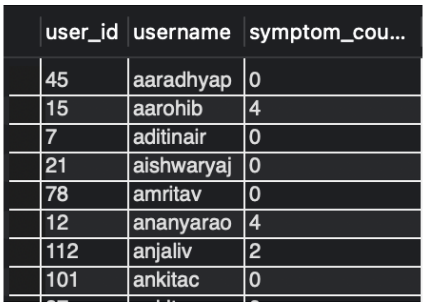
```
SELECT u.user_id, u.username
FROM User u
WHERE NOT EXISTS (
   SELECT 1
   FROM Flow f
   WHERE f.user_id = u.user_id
     AND f.flow_type = 'Heavy'
);
```



user_id	username
45	aaradhyap
7	aditinair
21	aishwaryaj
78	amritav
112	anjaliv
101	ankitac
87	ankitav
39	anushkad

Query 8: Find number of symptoms reported per user.

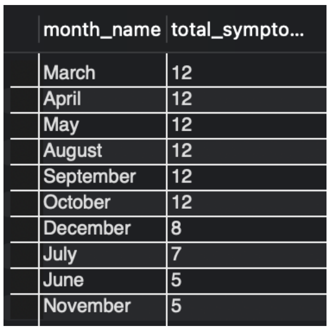
```
SELECT
   u.user_id,
   u.username,
   (SELECT COUNT(*) FROM Symptoms s WHERE s.user_id = u.user_id) AS
symptom_count
FROM User u;
```



user_id	username	symptom_cou...
45	aaradhyap	0
15	aarohib	4
7	aditinair	0
21	aishwaryaj	0
78	amritav	0
12	ananyarao	4
112	anjaliv	2
101	ankitac	0

Query 9: Find the number of symptoms reported per month using a derived table.

```
SELECT
   month_name,
   COUNT(*) AS total_symptoms
FROM (
   SELECT
      symptom_id,
      DATE_FORMAT(symptom_date, '%M') AS month_name
   FROM Symptoms
) AS monthly
```

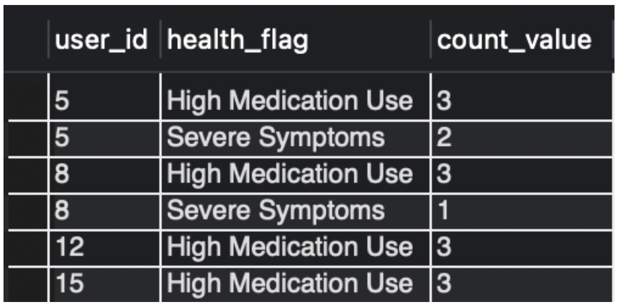


month_name	total_sympto...
March	12
April	12
May	12
August	12
September	12
October	12
December	8
July	7
June	5
November	5

```
GROUP BY month_name
ORDER BY total_symptoms DESC;
```

Query 10: Find users who either frequently take medications OR frequently log severe symptoms

```
-- Users with high medication usage
SELECT
   user_id,
   'High Medication Use' AS health_flag,
   COUNT(*) AS count_value
FROM Medication_Log
GROUP BY user_id
HAVING COUNT(*) >= 3

UNION

-- Users with severe symptoms
SELECT
   user_id,
   'Severe Symptoms' AS health_flag,
   COUNT(*) AS count_value
FROM Symptoms
WHERE severity = 'Severe'
GROUP BY user_id
HAVING COUNT(*) >= 1
ORDER BY user_id;
```

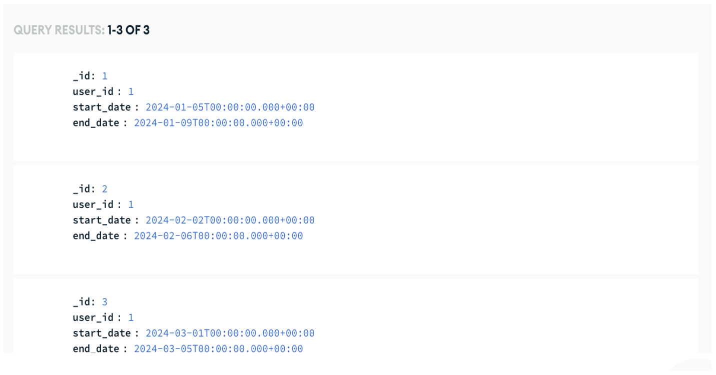


user_id	health_flag	count_value
5	High Medication Use	3
5	Severe Symptoms	2
8	High Medication Use	3
8	Severe Symptoms	1
12	High Medication Use	3
15	High Medication Use	3

NoSQL Implementation:

Query 1: Find period record entries for a user

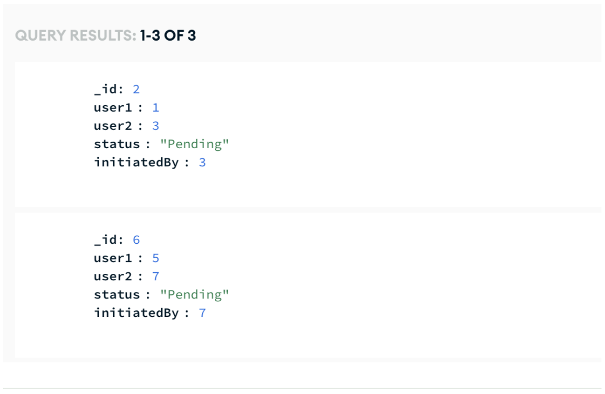
```
db.Period_record.find({ user_id: 1 })
```



```
QUERY RESULTS: 1-3 OF 3

_id: 1
user_id : 1
start_date : 2024-01-05T00:00:00.000+00:00
end_date : 2024-01-09T00:00:00.000+00:00

_id: 2
user_id : 1
start_date : 2024-02-02T00:00:00.000+00:00
end_date : 2024-02-06T00:00:00.000+00:00

_id: 3
user_id : 1
start_date : 2024-03-01T00:00:00.000+00:00
end_date : 2024-03-05T00:00:00.000+00:00
```

Query 2: Find pending friend requests initiated by a user

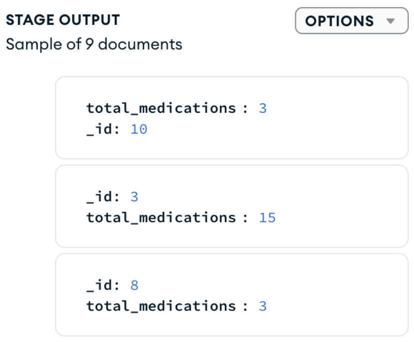
```
db.Friendship.find({
    status: "Pending",
    initiatedBy: { $exists: true }
})
```



```
QUERY RESULTS: 1-3 OF 3

_id: 2
user1 : 1
user2 : 3
status : "Pending"
initiatedBy : 3

_id: 6
user1 : 5
user2 : 7
status : "Pending"
initiatedBy : 7
```

Query 3: Find number of Medications taken per user

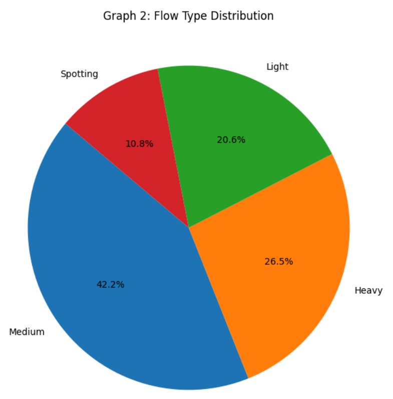
```
db.Medication_Log.aggregate([
{ $group: { _id: "$user_id", total_medications: { $sum: 1 } } }
])
```

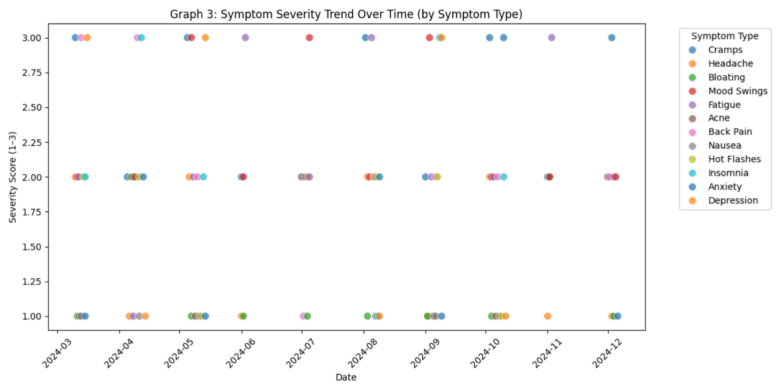


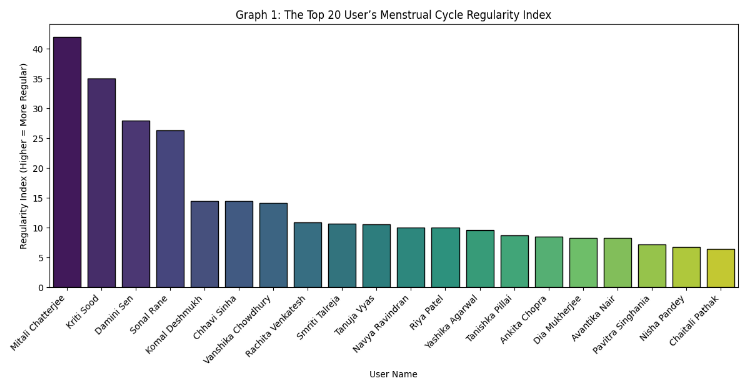
```
Sample of 9 documents

total_medications : 3
_id: 10

_id: 3
total_medications : 15

_id: 8
total_medications : 3
```

V. Database Access via Python

The database is accessed through Python using the mysql.connector library. After establishing the connection, SQL queries are executed using cursor.execute(), and the returned results are retrieved with cursor.fetchall().
The fetched data, initially in list format, is then converted into a Pandas DataFrame to enable efficient data manipulation and analysis.
For visualization, Matplotlib is used to generate analytical plots, allowing clear representation of trends, distributions, and relationships within the dataset. This workflow ensures a smooth pipeline from data extraction to meaningful visual insights.







VI. Summary and Recommendation

The Personalized Period Tracker with Analytics and Social Sync ended up becoming a well-rounded system that actually organizes menstrual health data in a way that feels practical instead of just technical. It keeps track of cycles, symptoms, flow patterns, and medications, and the social sync feature adds something most apps don't even try to offer. The database design holds everything together cleanly, making it possible to spot patterns, predict upcoming cycles with better accuracy, and run analytical queries without the data getting messy. Because the whole thing was built from a clear conceptual model and then turned into a proper relational design before being implemented in MySQL and tested with realistic sample data, it shows that the approach is not only workable but reliable.

There's definitely room to build on the system, adding things like mood, temperature, or weight over time could give users a clearer sense of how their cycle connects to what they feel day to day. It would also be important to improve privacy and consent options, since this kind of information is very personal. Including some basic medical context for conditions like PCOS or endometriosis, letting the system sync across different devices, and experimenting with machine learning for better predictions would all help move it beyond a simple class project. Changes like these would make the tool feel more reliable and better suited for a wider range of users.