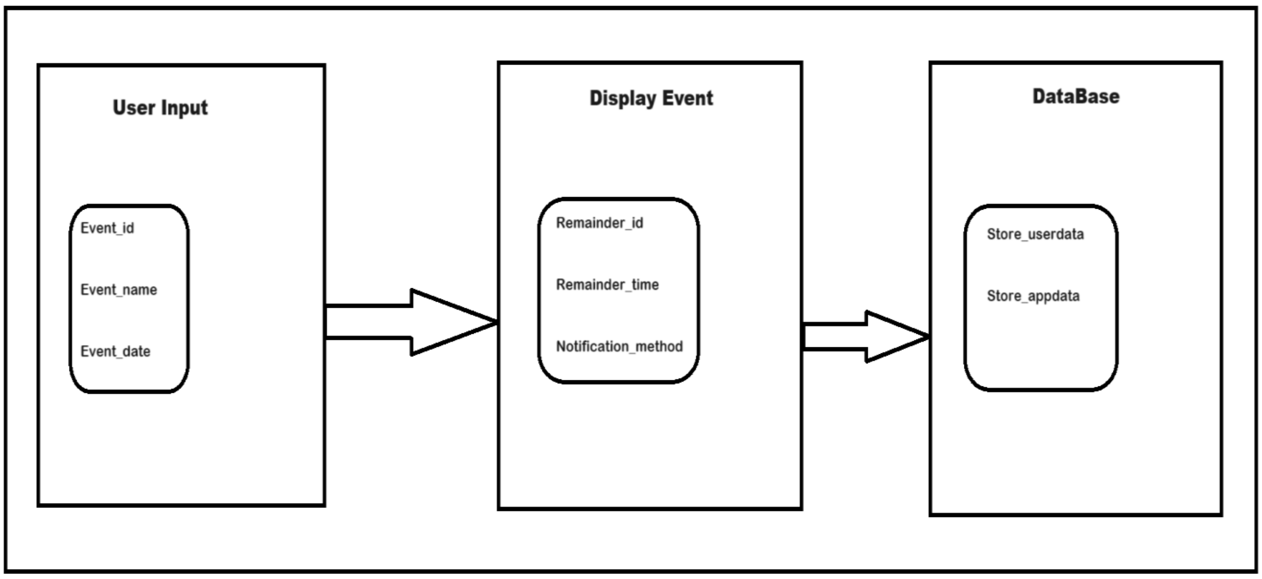
Database Schema Design

The database schema for the Event Management System is designed to effectively manage and store the data required for organizing and executing events.

Tables

Event_id	VARCHAR (100)	SERIAL PRIMARY KEY
Event_name	VARCHAR (100)	NOT NULL
Event_date	DATE	NOT NULL
Remainder_id	VARCHAR (100)	SERIAL PRIMARY KEY
Remainder_time	DATETIME	NOT NULL
Notification_method	ENUM (‘email’, ’SMS’, ’push_notification’)	NOT NULL

ER DIAGRAM:



2. Database Implementation

SQL Code:

```
-- Create Users Table
CREATE TABLE Users (
   user_id INT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(50) UNIQUE NOT NULL,
   password VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL,
   role ENUM('admin', 'organizer', 'participant') NOT NULL
);

-- Create Events Table
CREATE TABLE Events (
   event_id INT AUTO_INCREMENT PRIMARY KEY,
   event_name VARCHAR(100) NOT NULL,
   event_date DATE NOT NULL,
   event_location VARCHAR(255) NOT NULL,
   organizer_id INT,
   FOREIGN KEY (organizer_id) REFERENCES Users(user_id)
);

-- Create Tasks Table
CREATE TABLE Tasks (
   task_id INT AUTO_INCREMENT PRIMARY KEY,
   task_name VARCHAR(100) NOT NULL,
   task_description TEXT,
   assigned_to INT,
   event_id INT,
   status ENUM('pending', 'completed', 'in-progress') NOT NULL,
   FOREIGN KEY (assigned_to) REFERENCES Users(user_id),
   FOREIGN KEY (event_id) REFERENCES Events(event_id)
);

-- Create Resources Table
CREATE TABLE Resources (
   resource_id INT AUTO_INCREMENT PRIMARY KEY,
   resource_name VARCHAR(100) NOT NULL,
   resource_type VARCHAR(50) NOT NULL,
   event_id INT,
   availability ENUM('available', 'unavailable') NOT NULL,
   FOREIGN KEY (event_id) REFERENCES Events(event_id)
);
```

```
-- Create Payments Table
CREATE TABLE Payments (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    event_id INT,
    amount DECIMAL(10, 2) NOT NULL,
    payment_date DATE NOT NULL,
    payment_status ENUM('pending', 'completed', 'failed') NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (event_id) REFERENCES Events(event_id)
);

-- Insert Sample Data
INSERT INTO Users (username, password, email, role) VALUES
('john_doe', 'password123', 'john@example.com', 'organizer'),
('jane_smith', 'password456', 'jane@example.com', 'participant'),
('admin_user', 'adminpass', 'admin@example.com', 'admin');

INSERT INTO Events (event_name, event_date, event_location, organizer_id)
VALUES
('Annual Conference', '2024-09-15', 'Convention Center', 1),
('Team Building Workshop', '2024-10-01', 'Community Hall', 1);

INSERT INTO Tasks (task_name, task_description, assigned_to, event_id, status)
VALUES
('Book Venue', 'Reserve the conference hall', 1, 1, 'completed'),
('Send Invites', 'Email invitations to all participants', 2, 1, 'pending');

INSERT INTO Resources (resource_name, resource_type, event_id, availability)
VALUES
('Projector', 'Equipment', 1, 'available'),
('Microphone', 'Equipment', 1, 'available');

INSERT INTO Payments (user_id, event_id, amount, payment_date, payment_status)
VALUES
(2, 1, 100.00, '2024-08-15', 'completed'),
(2, 2, 50.00, '2024-08-16', 'pending');
```

3.Data Manipulation and Querying:

Insertion Queries:

```
-- Insert a new event
INSERT INTO Events (event_name, event_date, event_location, organizer_id)
VALUES ('Networking Event', '2024-11-01', 'City Hall', 1);

-- Insert a new task
INSERT INTO Tasks (task_name, task_description, assigned_to, event_id, status)
VALUES ('Order Catering', 'Arrange for catering services', 2, 2, 'in-progress');
```

Update Queries:

```
-- Update task status
UPDATE Tasks
SET status = 'completed'
WHERE task_id = 1;

-- Update payment status
UPDATE Payments
SET payment_status = 'completed'
WHERE payment_id = 2;
```

Deletion Queries:

```
-- Delete a user
DELETE FROM Users
WHERE user_id = 3;

-- Delete an event
DELETE FROM Events
WHERE event_id = 2;
```

Retrieval Queries:

1. **Join Query**:

```
-- Retrieve all tasks for a specific event
SELECT t.task_name, t.task_description, t.status
```

2. **Grouping and Aggregation**:

```
-- Get the total amount of payments per event
SELECT e.event_name,
FROM Payments p
JOIN Events e ON p.event_id = e.event_id
GROUP BY e.event_name;
```

3. **Subquery**:

```
-- Retrieve users who have made booking for events
SELECT username
FROM Users
WHERE user_id IN (SELECT DISTINCT user_id);
```

4. **Complex Query with Subquery**:

```
-- Get events with the highest number of tasks
SELECT e.event_name, COUNT(t.task_id) as task_count
FROM Events e, LEFT JOIN Tasks t ON e.event_id = t.event_id
```