# PECH Prayer Diary - Database Schema Enhancements

Based on my review of the PECH Prayer Diary codebase, I've identified opportunities to enhance the database schema to support additional features and improve data organization. This document outlines proposed schema changes and explains their benefits.

## Current Database Structure

From analyzing the code, the current database appears to have these main tables:

1. **profiles** - User information and permissions
2. **prayer_updates** - Weekly prayer updates
3. **prayer_topics** - Non-member prayer subjects
4. **urgent_prayers** - Time-sensitive prayer requests

## Proposed Schema Enhancements

### 1. Extended User Profile Schema

The current `profiles` table could benefit from additional fields to support enhanced notification preferences and usage statistics.

```sql
-- Add columns to profiles table
ALTER TABLE profiles ADD COLUMN notification_channels JSONB DEFAULT '{"email": true, "push": fa
ALTER TABLE profiles ADD COLUMN last_login TIMESTAMP WITH TIME ZONE;
ALTER TABLE profiles ADD COLUMN login_count INTEGER DEFAULT 0;
ALTER TABLE profiles ADD COLUMN theme_preference VARCHAR(20) DEFAULT 'light';
ALTER TABLE profiles ADD COLUMN calendar_display_type VARCHAR(20) DEFAULT 'large-photo';
```

### 2. User Groups Table

Add support for organizing users into groups (like family units, ministry teams, etc.) for better calendar organization.

```sql
-- Create user groups table
CREATE TABLE user_groups (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    created_by UUID REFERENCES profiles(id)
);

-- Create user group membership table
CREATE TABLE user_group_memberships (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    group_id UUID REFERENCES user_groups(id) ON DELETE CASCADE,
    is_admin BOOLEAN DEFAULT FALSE,
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    UNIQUE (user_id, group_id)
);

-- Add column to profiles for primary group
ALTER TABLE profiles ADD COLUMN primary_group_id UUID REFERENCES user_groups(id) ON DELETE SET
```

## 3. Enhanced Prayer Calendar Structure

Create a more flexible calendar assignment system:

```sql
-- Create calendar assignments table
CREATE TABLE calendar_assignments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    day_of_month INTEGER NOT NULL CHECK (day_of_month BETWEEN 1 AND 31),
    month_pattern INTEGER NOT NULL DEFAULT 0, -- 0 = all months, 1 = odd months, 2 = even month
    assignable_type VARCHAR(20) NOT NULL, -- 'user', 'group', or 'topic'
    assignable_id UUID NOT NULL, -- References profiles, user_groups, or prayer_topics
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    created_by UUID REFERENCES profiles(id),
    active BOOLEAN DEFAULT TRUE,
    UNIQUE (day_of_month, assignable_type, assignable_id)
);

-- Create index for fast calendar lookup
CREATE INDEX idx_calendar_assignments_day ON calendar_assignments(day_of_month) WHERE active =
```

## 4. Prayer Request Tracking

Add a system to track personal prayer requests with privacy controls:

```sql
-- Create prayer requests table
CREATE TABLE prayer_requests (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    content TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    answer_status VARCHAR(20) DEFAULT 'pending', -- 'pending', 'in_progress', 'answered'
    answer_notes TEXT,
    answered_at TIMESTAMP WITH TIME ZONE,
    visibility VARCHAR(20) DEFAULT 'private', -- 'private', 'group', 'public'
    group_id UUID REFERENCES user_groups(id) ON DELETE SET NULL,
    is_anonymous BOOLEAN DEFAULT FALSE
);

-- Create prayer request updates table
CREATE TABLE prayer_request_updates (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    prayer_request_id UUID REFERENCES prayer_requests(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    created_by UUID REFERENCES profiles(id)
);

-- Create prayer commitment table
CREATE TABLE prayer_commitments (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    prayer_request_id UUID REFERENCES prayer_requests(id) ON DELETE CASCADE,
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    committed_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    reminder_frequency VARCHAR(20) DEFAULT 'daily', -- 'daily', 'weekly', 'monthly'
    next_reminder_at TIMESTAMP WITH TIME ZONE,
    UNIQUE (prayer_request_id, user_id)
);
```

## 5. Notifications Table

Create a centralized notification system:

```sql
-- Create notifications table
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    message TEXT NOT NULL,
    notification_type VARCHAR(50) NOT NULL, -- 'update', 'urgent', 'request', 'answer', etc.
    content_id UUID, -- References the related content (prayer_updates, urgent_prayers, etc.)
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    read_at TIMESTAMP WITH TIME ZONE
);

-- Create push subscriptions table
CREATE TABLE push_subscriptions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    subscription JSONB NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    last_used_at TIMESTAMP WITH TIME ZONE,
    is_active BOOLEAN DEFAULT TRUE,
    UNIQUE (user_id, subscription)
);

-- Create notification delivery tracking
CREATE TABLE notification_deliveries (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    notification_id UUID REFERENCES notifications(id) ON DELETE CASCADE,
    channel VARCHAR(20) NOT NULL, -- 'email', 'push', 'sms', 'whatsapp'
    status VARCHAR(20) DEFAULT 'pending', -- 'pending', 'sent', 'delivered', 'failed'
    sent_at TIMESTAMP WITH TIME ZONE,
    delivered_at TIMESTAMP WITH TIME ZONE,
    error_message TEXT,
    retry_count INTEGER DEFAULT 0,
    UNIQUE (notification_id, channel)
);
```

## 6. Improved Prayer Updates Schema

Enhance the prayer updates table with categories and reactions:

```sql
-- Add columns to prayer_updates table
ALTER TABLE prayer_updates ADD COLUMN category VARCHAR(50);
ALTER TABLE prayer_updates ADD COLUMN tags TEXT[];
ALTER TABLE prayer_updates ADD COLUMN view_count INTEGER DEFAULT 0;

-- Create update reactions table
CREATE TABLE update_reactions (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    update_id UUID REFERENCES prayer_updates(id) ON DELETE CASCADE,
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    reaction_type VARCHAR(20) NOT NULL, -- 'praying', 'praise', 'encouragement'
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    UNIQUE (update_id, user_id, reaction_type)
);
```

## 7. Event Calendar

Add support for prayer events:

sql

```sql
-- Create prayer events table
CREATE TABLE prayer_events (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    title VARCHAR(255) NOT NULL,
    description TEXT,
    location TEXT,
    start_time TIMESTAMP WITH TIME ZONE NOT NULL,
    end_time TIMESTAMP WITH TIME ZONE NOT NULL,
    created_by UUID REFERENCES profiles(id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    is_recurring BOOLEAN DEFAULT FALSE,
    recurrence_pattern JSONB, -- For recurring events
    is_online BOOLEAN DEFAULT FALSE,
    meeting_link TEXT,
    reminder_time INTEGER -- Minutes before event to send reminder
);

-- Create event attendees table
CREATE TABLE event_attendees (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    event_id UUID REFERENCES prayer_events(id) ON DELETE CASCADE,
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    response_status VARCHAR(20) DEFAULT 'pending', -- 'pending', 'attending', 'not_attending',
    responded_at TIMESTAMP WITH TIME ZONE,
    UNIQUE (event_id, user_id)
);
```

## 8. Scripture References

Add a table to track scripture references:

```sql
-- Create scripture references table
CREATE TABLE scripture_references (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    book VARCHAR(50) NOT NULL,
    chapter INTEGER NOT NULL,
    verse_start INTEGER NOT NULL,
    verse_end INTEGER,
    reference_text TEXT, -- The actual scripture text
    reference_type VARCHAR(20) NOT NULL, -- 'update', 'urgent', 'topic', 'request'
    reference_id UUID NOT NULL, -- ID of the related content
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Create index for fast lookup
CREATE INDEX idx_scripture_references_type_id ON scripture_references(reference_type, reference
```

### 9. User Activity Logging

Add activity tracking for usage analytics:

```sql
-- Create user activity table
CREATE TABLE user_activities (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES profiles(id) ON DELETE CASCADE,
    activity_type VARCHAR(50) NOT NULL, -- 'login', 'view_update', 'create_request', etc.
    activity_data JSONB, -- Additional context about the activity
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    ip_address VARCHAR(45),
    user_agent TEXT
);

-- Create index for user activity queries
CREATE INDEX idx_user_activities_user_type ON user_activities(user_id, activity_type);
CREATE INDEX idx_user_activities_created_at ON user_activities(created_at);
```

## Database Triggers and Functions

To automate certain processes, these triggers and functions would be helpful:

### 1. Update Last Activity Trigger

```sql
-- Create function to update last activity
CREATE OR REPLACE FUNCTION update_last_activity()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE profiles
    SET last_active_at = NOW()
    WHERE id = NEW.user_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create trigger on user_activities table
CREATE TRIGGER trigger_update_last_activity
AFTER INSERT ON user_activities
FOR EACH ROW
EXECUTE FUNCTION update_last_activity();
```

## 2. Auto-Create Calendar Assignment on Profile Create

sql

```sql
-- Create function to assign new users to a calendar day
CREATE OR REPLACE FUNCTION assign_new_user_to_calendar()
RETURNS TRIGGER AS $$
DECLARE
    least_used_day INTEGER;
BEGIN
    -- Skip if calendar_hide is true
    IF NEW.calendar_hide = TRUE THEN
        RETURN NEW;
    END IF;

    -- Find the day with the fewest assignments
    SELECT day_of_month INTO least_used_day
    FROM (
        SELECT d.day AS day_of_month, COUNT(ca.id) AS assignment_count
        FROM generate_series(1, 31) AS d(day)
        LEFT JOIN calendar_assignments ca ON ca.day_of_month = d.day AND ca.active = TRUE
        GROUP BY d.day
        ORDER BY assignment_count ASC, d.day ASC
        LIMIT 1
    ) AS least_used;

    -- Create a calendar assignment for the new user
    INSERT INTO calendar_assignments (
        day_of_month,
        month_pattern,
        assignable_type,
        assignable_id,
        created_by
    ) VALUES (
        least_used_day,
        0, -- All months
        'user',
        NEW.id,
        NEW.id
    );

    -- Update the user's pray_day field
    UPDATE profiles
    SET pray_day = least_used_day
    WHERE id = NEW.id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```sql
-- Create trigger on profiles table
CREATE TRIGGER trigger_assign_new_user_to_calendar
AFTER INSERT ON profiles
FOR EACH ROW
WHEN (NEW.approval_state = 'Approved')
EXECUTE FUNCTION assign_new_user_to_calendar();
```

## 3. Notification Creation Function

sql

```sql
-- Create function to send notifications
CREATE OR REPLACE FUNCTION create_notification(
    p_user_id UUID,
    p_title VARCHAR(255),
    p_message TEXT,
    p_notification_type VARCHAR(50),
    p_content_id UUID
)
RETURNS UUID AS $$
DECLARE
    notification_id UUID;
BEGIN
    -- Insert the notification
    INSERT INTO notifications (
        user_id,
        title,
        message,
        notification_type,
        content_id
    ) VALUES (
        p_user_id,
        p_title,
        p_message,
        p_notification_type,
        p_content_id
    )
    RETURNING id INTO notification_id;

    -- Get user notification preferences
    DECLARE
        user_prefs JSONB;
    BEGIN
        SELECT notification_channels INTO user_prefs
        FROM profiles
        WHERE id = p_user_id;

        -- Create delivery records based on preferences
        IF user_prefs->>'email' = 'true' THEN
            INSERT INTO notification_deliveries (notification_id, channel, status)
            VALUES (notification_id, 'email', 'pending');
        END IF;

        IF user_prefs->>'push' = 'true' THEN
            INSERT INTO notification_deliveries (notification_id, channel, status)
            VALUES (notification_id, 'push', 'pending');
        END IF;
        IF user_prefs->>'sms' = 'true' THEN
            INSERT INTO notification_deliveries (notification_id, channel, status)
            VALUES (notification_id, 'sms', 'pending');
        END IF;

        IF user_prefs->>'whatsapp' = 'true' THEN
            INSERT INTO notification_deliveries (notification_id, channel, status)
            VALUES (notification_id, 'whatsapp', 'pending');
        END IF;
    END;

    RETURN notification_id;
END;
$$ LANGUAGE plpgsql;
```

## Migration Strategy

To implement these schema changes safely:

1. **Create Backup**: Take a full backup of the current database
2. **Staged Rollout**: Implement changes in stages:
   - Add new tables first
   - Add new columns to existing tables
   - Create database functions and triggers
   - Migrate existing data to new structures
3. **Data Migration**: Write scripts to migrate existing data:
   - Move pray_day and pray_months from profiles to calendar_assignments
   - Set up default notification preferences
   - Create initial user groups if needed
4. **Application Updates**: Update application code to use new schema in parallel with supporting legacy schema
5. **Test Thoroughly**: Test all functionality with the new schema before final deployment

## Benefits of the Enhanced Schema

These schema enhancements would provide several benefits:

1. **Better Organization**: More structured data relationships
2. **Enhanced Features**: Support for new features like prayer groups, event management, and reaction tracking

3. **Improved Notifications**: More reliable and trackable notification system
4. **Better Analytics**: User activity tracking for feature usage insights
5. **Scripture Integration**: Proper tracking of Bible references
6. **Scalability**: More flexible design that can accommodate future feature additions

## Example Queries With New Schema

Here are some examples of how the new schema enables powerful queries:

### Find all prayer assignments for a specific day and month type

```sql
SELECT
    a.day_of_month,
    a.assignable_type,
    CASE
        WHEN a.assignable_type = 'user' THEN p.full_name
        WHEN a.assignable_type = 'topic' THEN t.topic_title
        WHEN a.assignable_type = 'group' THEN g.name
    END AS name,
    CASE
        WHEN a.assignable_type = 'user' THEN p.profile_image_url
        WHEN a.assignable_type = 'topic' THEN t.topic_image_url
        WHEN a.assignable_type = 'group' THEN NULL
    END AS image_url,
    CASE
        WHEN a.assignable_type = 'user' THEN p.prayer_points
        WHEN a.assignable_type = 'topic' THEN t.topic_text
        WHEN a.assignable_type = 'group' THEN g.description
    END AS prayer_points
FROM
    calendar_assignments a
LEFT JOIN
    profiles p ON a.assignable_type = 'user' AND a.assignable_id = p.id
LEFT JOIN
    prayer_topics t ON a.assignable_type = 'topic' AND a.assignable_id = t.id
LEFT JOIN
    user_groups g ON a.assignable_type = 'group' AND a.assignable_id = g.id
WHERE
    a.day_of_month = 15
    AND (a.month_pattern = 0 OR a.month_pattern = 1) -- For odd months
    AND a.active = TRUE
ORDER BY
    a.assignable_type;
```

## Get all prayer requests with commitment counts

```sql
SELECT
    pr.id,
    pr.title,
    pr.content,
    pr.created_at,
    pr.answer_status,
    p.full_name AS requestor_name,
    CASE WHEN pr.is_anonymous THEN TRUE ELSE FALSE END AS is_anonymous,
    COUNT(pc.id) AS prayer_commitment_count
FROM
    prayer_requests pr
JOIN
    profiles p ON pr.user_id = p.id
LEFT JOIN
    prayer_commitments pc ON pr.id = pc.prayer_request_id
WHERE
    pr.visibility = 'public'
    OR (pr.visibility = 'group' AND pr.group_id IN (
        SELECT group_id FROM user_group_memberships WHERE user_id = '00000000-0000-0000-0000-00
    ))
GROUP BY
    pr.id, p.full_name
ORDER BY
    pr.created_at DESC;
```

## Find users with no recent activity

```sql
SELECT
    p.id,
    p.full_name,
    p.email,
    MAX(ua.created_at) AS last_activity
FROM
    profiles p
LEFT JOIN
    user_activities ua ON p.id = ua.user_id
WHERE
    p.approval_state = 'Approved'
GROUP BY
    p.id
HAVING
    MAX(ua.created_at) IS NULL OR MAX(ua.created_at) < NOW() - INTERVAL '30 days'
ORDER BY
    last_activity ASC NULLS FIRST;
```

## Conclusion

The proposed schema enhancements would significantly improve the PECH Prayer Diary application's data structure while enabling new features and better performance. By implementing these changes, the app can grow beyond its current capabilities while maintaining a solid foundation for future development.

These improvements align well with the app's core purpose of managing prayer ministry while adding new capabilities for community engagement, event coordination, and personalized prayer experiences.