ImageFlow - Database Design

Author: Sinem Eissler
Date: July 18, 2025

---

1. Database Technology Choice: PostgreSQL (Relational Database)

Justification for PostgreSQL

After analyzing ImageFlow's complex requirements involving social networking, collaboration, and version control, I have selected PostgreSQL as the primary database technology. This choice is driven by several critical factors:

1. Complex Relationships

- Many-to-many relationships (users following users, shared editing sessions)
- Hierarchical data (image versions with branching)
- Social graph queries requiring joins
- Permission systems with cascading rules

2. ACID Compliance

- Critical for maintaining data integrity in collaborative editing
- Ensures consistency in social interactions (follows, likes)
- Reliable transaction support for complex operations

3. Advanced Features

- JSONB support for flexible metadata storage
- Full-text search capabilities for custom search engine
- Window functions for activity feed ranking
- CTEs for efficient recursive queries (version trees)

4. Custom Implementation Benefits

- Direct SQL control allows optimization of complex queries
- Stored procedures for performance-critical operations
- Custom indexing strategies for specific access patterns
- Trigger-based audit trails and notifications

5. Scalability Path

- Read replicas for scaling read-heavy operations
- Partitioning for large tables (images, activities)
- Connection pooling for efficient resource usage

---

2. Entity-Relationship Diagram Overview

The database design implements a normalized structure with strategic denormalization for performance. The schema supports all custom features including the canvas editor, social networking, and version control.

---

3. Database Tables and Relationships

Users Table

```sql
CREATE TABLE users (
  user_id VARCHAR(20) PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  display_name VARCHAR(100) NOT NULL,
  bio TEXT,
  avatar_url VARCHAR(500),
  email_verified BOOLEAN DEFAULT FALSE,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
  last_login_at TIMESTAMP WITH TIME ZONE,
  settings JSONB DEFAULT '{}',

  INDEX idx_users_username (username),
  INDEX idx_users_email (email),
  INDEX idx_users_created_at (created_at)
);
```

Purpose: Core user authentication and profile information. The settings JSONB field stores user preferences for the canvas editor and notification settings. This table is accessed for every authenticated request.

Relationships:

- One-to-Many with images (user owns many images)
- Many-to-Many with users through follows table (social graph)
- One-to-Many with sessions (user has many login sessions)

Images Table

```sql
CREATE TABLE images (
  image_id VARCHAR(20) PRIMARY KEY,
  user_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  filename VARCHAR(255) NOT NULL,
  mime_type VARCHAR(50) NOT NULL,
  file_size INTEGER NOT NULL,
  width INTEGER NOT NULL,
  height INTEGER NOT NULL,
  storage_path VARCHAR(500) NOT NULL,
  thumbnail_paths JSONB NOT NULL,
  privacy_level VARCHAR(20) DEFAULT 'public',
  is_deleted BOOLEAN DEFAULT FALSE,
  uploaded_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
  last_edited_at TIMESTAMP WITH TIME ZONE,
  metadata JSONB DEFAULT '{}',
  color_palette JSONB,
  ai_analysis JSONB,
  view_count INTEGER DEFAULT 0,

  INDEX idx_images_user_id (user_id),
  INDEX idx_images_uploaded_at (uploaded_at),
  INDEX idx_images_privacy_level (privacy_level),
  FULLTEXT INDEX idx_images_fulltext (title, description)
);
```

Purpose: Stores image metadata and processing results. The JSONB fields hold flexible data like EXIF information, color analysis results, and AI-generated labels. This is the central table for the gallery and image management features.

Relationships:

- Many-to-One with users (image belongs to one user)
- One-to-Many with image_versions (image has many versions)
- One-to-Many with image_tags (image has many tags)
- One-to-Many with comments (image has many comments)

Image Versions Table

```sql
CREATE TABLE image_versions (
    version_id VARCHAR(20) PRIMARY KEY,
    image_id VARCHAR(20) NOT NULL REFERENCES images(image_id) ON DELETE CASCADE,
    parent_version_id VARCHAR(20) REFERENCES image_versions(version_id),
    version_number INTEGER NOT NULL,
    branch_name VARCHAR(100),
    created_by VARCHAR(20) NOT NULL REFERENCES users(user_id),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    description TEXT,
    canvas_state JSONB NOT NULL,
    thumbnail_path VARCHAR(500),
    file_size INTEGER NOT NULL,
    is_current BOOLEAN DEFAULT FALSE,

    INDEX idx_versions_image_id (image_id),
    INDEX idx_versions_parent (parent_version_id),
    INDEX idx_versions_created_at (created_at),
    UNIQUE INDEX idx_versions_image_number (image_id, version_number)
);
```

Purpose: Implements version control for images. Each version stores the complete canvas state including layers, transformations, and filters. Supports branching through parent_version_id self-reference.

Example Usage: When Sarah saves her edits, a new version is created with the current canvas state. She can later branch from any version to try experimental edits.

Canvas Layers Table

```sql
```

```
CREATE TABLE canvas_layers (
    layer_id VARCHAR(20) PRIMARY KEY,
    version_id VARCHAR(20) NOT NULL REFERENCES image_versions(version_id) ON DELETE CASCADE,
    layer_order INTEGER NOT NULL,
    layer_type VARCHAR(50) NOT NULL,
    name VARCHAR(100) NOT NULL,
    visible BOOLEAN DEFAULT TRUE,
    opacity DECIMAL(3,2) DEFAULT 1.0,
    blend_mode VARCHAR(50) DEFAULT 'normal',
    locked BOOLEAN DEFAULT FALSE,
    layer_data JSONB NOT NULL,
    transform_matrix JSONB,
    filters JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_layers_version_id (version_id),
    INDEX idx_layers_order (version_id, layer_order)
);
```

Purpose: Stores individual layers for the canvas editor. Each layer can be an image, adjustment, drawing, or text layer. The layer_data JSONB contains type-specific information.

Follows Table (Many-to-Many)

```sql
CREATE TABLE follows (
    follower_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    following_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    followed_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    notifications_enabled BOOLEAN DEFAULT TRUE,

    PRIMARY KEY (follower_id, following_id),
    INDEX idx_follows_following_id (following_id),
    INDEX idx_follows_followed_at (followed_at),
    CHECK (follower_id != following_id)
);
```

Purpose: Implements the social graph for the following system. This bidirectional relationship table enables efficient queries for both followers and following lists.

Example Usage: When Marcus follows Sarah, a record is created. The activity feed algorithm uses this table to determine which updates to show.

Activities Table

```sql
CREATE TABLE activities (
    activity_id VARCHAR(20) PRIMARY KEY,
    actor_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    activity_type VARCHAR(50) NOT NULL,
    target_type VARCHAR(50) NOT NULL,
    target_id VARCHAR(20) NOT NULL,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_activities_actor_id (actor_id),
    INDEX idx_activities_created_at (created_at),
    INDEX idx_activities_target (target_type, target_id),
    INDEX idx_activities_type (activity_type)
);
```

Purpose: Records all user activities for the social feed. The flexible schema supports different activity types (upload, edit, follow, comment) with type-specific metadata.

Comments Table

```sql
CREATE TABLE comments (
    comment_id VARCHAR(20) PRIMARY KEY,
    image_id VARCHAR(20) NOT NULL REFERENCES images(image_id) ON DELETE CASCADE,
    user_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    parent_comment_id VARCHAR(20) REFERENCES comments(comment_id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    coordinates JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    edited_at TIMESTAMP WITH TIME ZONE,
    is_deleted BOOLEAN DEFAULT FALSE,

    INDEX idx_comments_image_id (image_id),
    INDEX idx_comments_user_id (user_id),
    INDEX idx_comments_created_at (created_at)
);
```

Purpose: Stores comments with optional image coordinates for contextual feedback. Supports nested replies through parent_comment_id.

Likes Table

sql

```sql
CREATE TABLE likes (
    user_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    target_type VARCHAR(50) NOT NULL,
    target_id VARCHAR(20) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    PRIMARY KEY (user_id, target_type, target_id),
    INDEX idx_likes_target (target_type, target_id),
    INDEX idx_likes_created_at (created_at)
);
```

Purpose: Polymorphic likes table supporting likes on images, comments, and other content types.

Edit Sessions Table

sql

```sql
CREATE TABLE edit_sessions (
    session_id VARCHAR(20) PRIMARY KEY,
    image_id VARCHAR(20) NOT NULL REFERENCES images(image_id) ON DELETE CASCADE,
    owner_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP WITH TIME ZONE NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    session_type VARCHAR(50) DEFAULT 'solo',
    settings JSONB DEFAULT '{}',

    INDEX idx_sessions_image_id (image_id),
    INDEX idx_sessions_owner_id (owner_id),
    INDEX idx_sessions_expires_at (expires_at)
);
```

Purpose: Manages collaborative editing sessions with expiration and access control.

Session Participants Table

sql

```
CREATE TABLE session_participants (
    session_id VARCHAR(20) NOT NULL REFERENCES edit_sessions(session_id) ON DELETE CASCADE,
    user_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    joined_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    last_active_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    permissions JSONB DEFAULT '{"canEdit": true, "canComment": true}',
    cursor_color VARCHAR(7),

    PRIMARY KEY (session_id, user_id),
    INDEX idx_participants_user_id (user_id)
);
```

Purpose: Tracks participants in collaborative editing sessions with their permissions and presence information.

Tags Table

```sql
CREATE TABLE tags (
    tag_id VARCHAR(20) PRIMARY KEY,
    tag_name VARCHAR(50) UNIQUE NOT NULL,
    tag_slug VARCHAR(50) UNIQUE NOT NULL,
    usage_count INTEGER DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_tags_name (tag_name),
    INDEX idx_tags_usage (usage_count DESC)
);
```

Purpose: Master list of all tags in the system with usage statistics for trending calculations.

Image Tags Table (Many-to-Many)

```sql
```

```
CREATE TABLE image_tags (
    image_id VARCHAR(20) NOT NULL REFERENCES images(image_id) ON DELETE CASCADE,
    tag_id VARCHAR(20) NOT NULL REFERENCES tags(tag_id) ON DELETE CASCADE,
    added_by VARCHAR(20) NOT NULL REFERENCES users(user_id),
    added_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    source VARCHAR(20) DEFAULT 'user',
    confidence DECIMAL(3,2),

    PRIMARY KEY (image_id, tag_id),
    INDEX idx_image_tags_tag_id (tag_id),
    INDEX idx_image_tags_added_at (added_at)
);
```

Purpose: Links images to tags with metadata about who added them and confidence scores for AI-generated tags.

Notifications Table

```sql
CREATE TABLE notifications (
    notification_id VARCHAR(20) PRIMARY KEY,
    user_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    type VARCHAR(50) NOT NULL,
    actor_id VARCHAR(20) REFERENCES users(user_id),
    target_type VARCHAR(50),
    target_id VARCHAR(20),
    message TEXT NOT NULL,
    data JSONB DEFAULT '{}',
    read_at TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_notifications_user_id (user_id),
    INDEX idx_notifications_unread (user_id, read_at),
    INDEX idx_notifications_created_at (created_at)
);
```

Purpose: Stores user notifications with flexible data field for type-specific information.

Search Index Table

```sql
```

```sql
CREATE TABLE search_index (
    index_id SERIAL PRIMARY KEY,
    document_type VARCHAR(50) NOT NULL,
    document_id VARCHAR(20) NOT NULL,
    content TEXT NOT NULL,
    metadata JSONB DEFAULT '{}',
    tsv_content TSVECTOR,
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,

    UNIQUE INDEX idx_search_document (document_type, document_id),
    INDEX idx_search_tsv USING GIN(tsv_content)
);
```

Purpose: Custom search index using PostgreSQL's full-text search capabilities. Updated via triggers when content changes.

User Sessions Table

```sql
CREATE TABLE user_sessions (
    session_id VARCHAR(20) PRIMARY KEY,
    user_id VARCHAR(20) NOT NULL REFERENCES users(user_id) ON DELETE CASCADE,
    refresh_token_hash VARCHAR(255) UNIQUE NOT NULL,
    device_info JSONB,
    ip_address INET,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    last_active_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    expires_at TIMESTAMP WITH TIME ZONE NOT NULL,

    INDEX idx_sessions_user_id (user_id),
    INDEX idx_sessions_expires_at (expires_at)
);
```

Purpose: Manages user authentication sessions with device tracking and refresh token storage.

---

4. Database Views for Performance

User Stats View

```sql
```

```
CREATE MATERIALIZED VIEW user_stats AS
SELECT
    u.user_id,
    COUNT(DISTINCT i.image_id) as image_count,
    COUNT(DISTINCT f1.follower_id) as follower_count,
    COUNT(DISTINCT f2.following_id) as following_count,
    SUM(i.view_count) as total_views,
    MAX(i.uploaded_at) as last_upload
FROM users u
LEFT JOIN images i ON u.user_id = i.user_id
LEFT JOIN follows f1 ON u.user_id = f1.following_id
LEFT JOIN follows f2 ON u.user_id = f2.follower_id
GROUP BY u.user_id;

CREATE INDEX idx_user_stats_user_id ON user_stats(user_id);
```

Purpose: Pre-calculated user statistics to avoid expensive joins on every profile view.

Activity Feed View

```sql
CREATE VIEW activity_feed AS
SELECT
    a.*,
    u.username as actor_username,
    u.display_name as actor_display_name,
    u.avatar_url as actor_avatar
FROM activities a
JOIN users u ON a.actor_id = u.user_id;
```

Purpose: Simplified view for building activity feeds with user information.

---

5. Custom Functions and Triggers

Update User Stats Trigger

```sql
```

```
CREATE OR REPLACE FUNCTION update_user_stats()
RETURNS TRIGGER AS $$
BEGIN
  -- Refresh materialized view for affected user
  REFRESH MATERIALIZED VIEW CONCURRENTLY user_stats
  WHERE user_id = NEW.user_id OR user_id = OLD.user_id;
  RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_stats
AFTER INSERT OR UPDATE OR DELETE ON images
FOR EACH ROW EXECUTE FUNCTION update_user_stats();
```

Search Index Update Trigger

sql

```
CREATE OR REPLACE FUNCTION update_search_index()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO search_index (document_type, document_id, content, tsv_content)
  VALUES (
    'image',
    NEW.image_id,
    NEW.title || ' ' || COALESCE(NEW.description, ''),
    to_tsvector('english', NEW.title || ' ' || COALESCE(NEW.description, ''))
  )
  ON CONFLICT (document_type, document_id) DO UPDATE
  SET content = EXCLUDED.content,
    tsv_content = EXCLUDED.tsv_content,
    updated_at = CURRENT_TIMESTAMP;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

---

6. Indexes and Performance Optimization

Composite Indexes for Common Queries

sql

```
-- Gallery view optimization
CREATE INDEX idx_images_gallery
ON images(user_id, uploaded_at DESC)
WHERE is_deleted = FALSE;

-- Activity feed optimization
CREATE INDEX idx_activities_feed
ON activities(actor_id, created_at DESC);

-- Tag search optimization
CREATE INDEX idx_image_tags_search
ON image_tags(tag_id, image_id);
```

Partial Indexes for Efficiency

```sql
-- Active sessions only
CREATE INDEX idx_active_sessions
ON edit_sessions(expires_at)
WHERE is_active = TRUE;

-- Unread notifications
CREATE INDEX idx_unread_notifications
ON notifications(user_id, created_at DESC)
WHERE read_at IS NULL;
```

---

7. Security and Access Control

Row-Level Security Policies

```sql
```

```
-- Users can only update their own profiles
ALTER TABLE users ENABLE ROW LEVEL SECURITY;

CREATE POLICY users_update_own
ON users FOR UPDATE
USING (user_id = current_user_id());

-- Images respect privacy settings
CREATE POLICY images_view_policy
ON images FOR SELECT
USING (
  privacy_level = 'public'
  OR user_id = current_user_id()
  OR EXISTS (
    SELECT 1 FROM follows
    WHERE follower_id = current_user_id()
    AND following_id = images.user_id
  )
);
```

Database Users and Permissions

```sql
-- Application user with limited permissions
CREATE USER imageflow_app WITH PASSWORD 'secure_password';
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO imageflow_app;
GRANT DELETE ON user_sessions, notifications TO imageflow_app;

-- Read-only user for analytics
CREATE USER imageflow_readonly WITH PASSWORD 'readonly_password';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO imageflow_readonly;
```

Conclusion

This PostgreSQL-based design provides a robust foundation for ImageFlow's complex features. The normalized structure with strategic denormalization ensures data integrity while maintaining performance. Custom functions, triggers, and views demonstrate advanced database programming skills beyond simple CRUD operations. The schema supports all custom features including the canvas editor, social networking, version control, and real-time collaboration.