Issue #2: PostgreSQL database schema and integration

Labels: `enhancement`, `database`, `infrastructure`
Milestone: Foundation Setup
Estimated Effort: 8-12 hours

Description

Replace the current file-based JSON storage with PostgreSQL database integration to enable data persistence, scalability, and concurrent access. This is critical for production deployment on Railway.com.

Problem Statement

Current implementation uses `leaderboard.json` file which:
- ❌ Loses data on container restart (ephemeral storage)
- ❌ Has race conditions with concurrent writes
- ❌ Cannot scale horizontally (multiple instances)
- ❌ No backup or recovery mechanism
- ❌ No transaction support

Tasks

1. Database Schema Design

- [ ] Create database migration files
- [ ] Design leaderboard table schema
- [ ] Add indexes for performance
- [ ] Add timestamps for audit trail

2. Database Client Integration

- [ ] Add `pg` (node-postgres) dependency
- [ ] Create database connection module
- [ ] Implement connection pooling
- [ ] Add environment variable for `DATABASE_URL`

3. Storage Layer Refactoring

- [ ] Replace `loadLeaderboard()` with database query
- [ ] Replace `saveLeaderboard()` with database upsert
- [ ] Add transaction support for atomic updates
- [ ] Implement error handling and retries

4. Migration Script

- [ ] Create script to migrate data from JSON to PostgreSQL
- [ ] Add validation to ensure data integrity
- [ ] Create rollback mechanism

5. Testing

- [ ] Add database unit tests
- [ ] Add integration tests with test database
- [ ] Test concurrent access scenarios
- [ ] Performance benchmarking

Database Schema

```
-- migrations/001_initial_schema.sql

-- Leaderboard table
CREATE TABLE IF NOT EXISTS leaderboard (
  user_id VARCHAR(20) PRIMARY KEY,
  score INTEGER DEFAULT 0 NOT NULL,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Index for leaderboard queries (sorted by score DESC)
CREATE INDEX idx_leaderboard_score ON leaderboard(score DESC);

-- Vote history table (optional, for analytics)
CREATE TABLE IF NOT EXISTS vote_history (
  id SERIAL PRIMARY KEY,
  voter_id VARCHAR(20) NOT NULL,
  voted_user_id VARCHAR(20) NOT NULL,
  vote_type VARCHAR(2) NOT NULL CHECK (vote_type IN ('++', '--')),
  channel_id VARCHAR(20),
  message_ts VARCHAR(20),
  created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_vote_history_user ON vote_history(voted_user_id);
CREATE INDEX idx_vote_history_created ON vote_history(created_at DESC);
```

Code Implementation

Database Connection Module

```
// db.js
const { Pool } = require('pg');

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: process.env.NODE_ENV === 'production'
    ? { rejectUnauthorized: false }
    : false,
  max: 10, // Maximum connection pool size
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

// Test connection on startup
pool.on('connect', () => {
  console.log('✓ Database connected');
});

pool.on('error', (err) => {
  console.error('✗ Unexpected database error:', err);
  process.exit(-1);
});

module.exports = pool;
```

Updated Storage Functions

```
// storage.js
const pool = require('./db');

/**
 * Get user's current score
 */
async function getUserScore(userId) {
  const result = await pool.query(
    'SELECT score FROM leaderboard WHERE user_id = $1',
    [userId]
  );
  return result.rows[0]?.score || 0;
}

/**
 * Update user score (atomic operation)
 */
async function updateUserScore(userId, delta) {
  const result = await pool.query(
    `INSERT INTO leaderboard (user_id, score, updated_at)
     VALUES ($1, $2, NOW())
     ON CONFLICT (user_id)
     DO UPDATE SET
       score = leaderboard.score + $2,
       updated_at = NOW()
     RETURNING score`,
    [userId, delta]
  );
  return result.rows[0].score;
}

/**
 * Get top N users for leaderboard
 */
async function getTopUsers(limit = 10) {
  const result = await pool.query(
    `SELECT user_id, score
     FROM leaderboard
     WHERE score != 0
     ORDER BY score DESC
     LIMIT $1`,
    [limit]
  );
  return result.rows;
}

/**
 * Record vote in history (optional)
 */
async function recordVote(voterId, votedUserId, voteType, channelId, messageTs) {
  await pool.query(
    `INSERT INTO vote_history
     (voter_id, voted_user_id, vote_type, channel_id, message_ts)
     VALUES ($1, $2, $3, $4, $5)`,
    [voterId, votedUserId, voteType, channelId, messageTs]
  );
}

module.exports = {
  getUserScore,
  updateUserScore,
  getTopUsers,
```

```
  recordVote,
  pool, // Export for graceful shutdown
};
```

Migration Script

```
// scripts/migrate-to-postgres.js
const fs = require('fs');
const path = require('path');
const pool = require('../db');

async function migrateData() {
  try {
    // Load existing JSON data
    const jsonPath = path.join(__dirname, '..', 'leaderboard.json');
    if (!fs.existsSync(jsonPath)) {
      console.log('No existing leaderboard.json found, skipping migration');
      return;
    }

    const data = JSON.parse(fs.readFileSync(jsonPath, 'utf8'));
    console.log(`Found ${Object.keys(data).length} users to migrate`);

    // Migrate each user
    for (const [userId, score] of Object.entries(data)) {
      await pool.query(
        `INSERT INTO leaderboard (user_id, score)
         VALUES ($1, $2)
         ON CONFLICT (user_id) DO NOTHING`,
        [userId, score]
      );
      console.log(`✓ Migrated user ${userId} with score ${score}`);
    }

    console.log('✓ Migration complete');

    // Backup old file
    const backupPath = `${jsonPath}.backup.${Date.now()}`;
    fs.renameSync(jsonPath, backupPath);
    console.log(`✓ Backed up old data to ${backupPath}`);
  } catch (error) {
    console.error('✗ Migration failed:', error);
    throw error;
  } finally {
    await pool.end();
  }
}

migrateData();
```

Environment Variables

```
# Add to .env
DATABASE_URL=postgresql://user:password@localhost:5432/ppbot

# Railway.com will automatically provide this when PostgreSQL is added
```

Testing Strategy

Unit Tests

```
// storage.test.js
const { getUserScore, updateUserScore, getTopUsers } = require('./storage');
const pool = require('./db');

beforeAll(async () => {
  // Create test database schema
  await pool.query('DROP TABLE IF EXISTS leaderboard');
  await pool.query(`
    CREATE TABLE leaderboard (
      user_id VARCHAR(20) PRIMARY KEY,
      score INTEGER DEFAULT 0,
      created_at TIMESTAMP DEFAULT NOW(),
      updated_at TIMESTAMP DEFAULT NOW()
    )
  `);
});

afterAll(async () => {
  await pool.end();
});

describe('Database operations', () => {
  test('getUserScore returns 0 for new user', async () => {
    const score = await getUserScore('U12345');
    expect(score).toBe(0);
  });

  test('updateUserScore increments score', async () => {
    const newScore = await updateUserScore('U12345', 1);
    expect(newScore).toBe(1);
  });

  test('getTopUsers returns sorted list', async () => {
    await updateUserScore('U11111', 10);
    await updateUserScore('U22222', 5);
    await updateUserScore('U33333', 15);

    const top = await getTopUsers(3);
    expect(top[0].user_id).toBe('U33333');
    expect(top[0].score).toBe(15);
    expect(top[1].user_id).toBe('U11111');
    expect(top[2].user_id).toBe('U22222');
  });
});
```

Acceptance Criteria

- [ ] Database schema is properly designed and documented
- [ ] Connection pooling is configured correctly
- [ ] All file-based operations replaced with database queries
- [ ] Migration script successfully transfers existing data
- [ ] Comprehensive tests pass (unit + integration)
- [ ] Error handling and retries are implemented

- [ ] Performance is acceptable (< 100ms for typical operations)
- [ ] Documentation updated with database setup instructions
- [ ] Works with Railway.com PostgreSQL addon

Performance Targets

- Vote processing: < 100ms p95
- Leaderboard query: < 50ms p95
- Concurrent votes: Handle 10+ simultaneous requests

Resources

- node-postgres Documentation (https://node-postgres.com/)
- PostgreSQL Best Practices (https://wiki.postgresql.org/wiki/Don't_Do_This)
- Railway.com PostgreSQL Guide (https://docs.railway.app/databases/postgresql)

Dependencies

This issue blocks:
- Issue #4: Railway.com deployment
- Issue #6: Leaderboard viewing enhancements