SCALABLE MULTIPLAYER CARD GAME ARCHITECTURE

This document outlines a robust and efficient architecture for a multiplayer card game, emphasizing scalability, security, and maintainability. It leverages static tables linked via foreign keys, eliminating dynamic table creation. Game logic runs on a backend (e.g., Render), with state stored in a database (e.g., Supabase).

DATABASE SCHEMA

The system utilizes five static tables:

- **Games Table:** `id`, `status`, `trump`, `current_turn`, `created_at` - Core game state.
- **Players Table:** `id`, `game_id`, `user_id`, `wallet`, `is_host`, `score` - User-game association, individual scores.
- **Hands Table:** `id`, `game_id`, `player_id`, `cards` (JSONB), `rounds_won`, `score` - Player's cards and round performance.
- **Rounds Table:** `id`, `game_id`, `round_number`, `winner_id`, `points_awarded`, `created_at` - Outcome and details of each round.
- **Moves Table:** `id`, `game_id`, `player_id`, `card` (JSONB), `played_at` - Log of every card played.

IMPROVED GAME FLOW

The game flow is structured into key stages:

1. **Setup:** Host creates game; players join. Backend enables "Continue" for host upon required player count (e.g., 4).
2. **Start Game:** Host action triggers backend shuffle & deal; `Hands` table updated.
3. **Gameplay Loop:** Backend validates plays (logged in `Moves`), determines round winner (updates `Rounds`), and adjusts `Hands` scores.
4. **End Game:** After 13 rounds, backend finalizes results, marks `Games` as 'finished'. Frontend displays summary; old data retained.

KEY ADVANTAGES

- **No Dynamic Tables:** Simplifies deployment; eliminates runtime schema changes.
- **High Scalability:** Static, indexed tables handle thousands of games efficiently.
- **Enhanced Security:** Supports robust Row-Level Security (RLS) policies.
- **Data Longevity:** Old game data available for leaderboards, analytics, auditing.