How We Work Together

When doing audits we only examine current codes and database information

WE DO NOT SIMPLY GO FROM OLD DOCUMENTATION

Its ok to say “I don’t know the answer”

no outside ai api's

we don’t make up answers that aren’t real

we don’t create situations that aren’t real

we use facts and figures and not superlatives

must use our hex colour code system that already exists using the hex generator code that we have

must use a mac friendly eof or sed terminal prompt with no # hashtag

descriptions to do this

no mock data or hardcoded answers ever

no simulations at all

all systems must be agnostic

we aim for the best solution not the fastest solutions

don’t assume anything about our code or database

we only examine full codes unless stated otherwise

Remember we have a DB terminal window for database enquiries and a terminal window for code

STAGE 1: Database Foundation (Universal Entities Table)

Step 1.1: Examine Current Database Structure

DB Terminal:

\dt

\d entities

\d character_profiles

\d knowledge_items

\d multiverse_events
Purpose: See what tables currently exist and their exact structure before making any changes.

Step 1.2: Check Current Hex ID Generator
Code Terminal:

cat backend/utils/hexIdGenerator.js
Purpose: Understand how your hex system works before using it in new schema.

Step 1.3: Check PostgreSQL Extensions Available
DB Terminal:

SELECT * FROM pg_available_extensions WHERE name IN ('pg_trgm', 'fuzzystrmatch');
Purpose: Verify we can use phonetic matching and fuzzy matching extensions.

Step 1.4: Design Universal Entities Table Schema
Based on all three recommendations + your hex system:

sql
```
CREATE TABLE entities (
    entity_hex_id VARCHAR(32) PRIMARY KEY,     -- Your hex system
    realm_hex_id VARCHAR(32) NOT NULL,         -- Instance isolation
    entity_type VARCHAR(30) NOT NULL,          -- PERSON, LOCATION, EVENT, TIME, OBJECT, CONCEPT
    entity_name VARCHAR(255) NOT NULL,
    entity_name_normalized VARCHAR(255) NOT NULL,

    -- Phonetic codes (precomputed)
```

```
    phonetic_soundex VARCHAR(20),
    phonetic_metaphone VARCHAR(20),
    phonetic_double_metaphone_1 VARCHAR(22),
    phonetic_double_metaphone_2 VARCHAR(22),

    -- Fuzzy matching
    aliases TEXT[],
    aliases_normalized TEXT[],

    -- Flexible metadata per realm
    metadata JSONB DEFAULT '{}',

    -- Description for WHAT questions
    description TEXT,

    -- Standard timestamps
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

Review this schema - does it match what we need?

Step 1.5: Create Indexes for Performance

```
sql
-- Critical: Always filter by realm first
CREATE INDEX idx_entities_realm_type ON entities(realm_hex_id, entity_type);
CREATE INDEX idx_entities_realm_name ON entities(realm_hex_id,
entity_name_normalized);

-- Phonetic search
CREATE INDEX idx_entities_soundex ON entities(phonetic_soundex);
CREATE INDEX idx_entities_metaphone ON entities(phonetic_metaphone);
CREATE INDEX idx_entities_dmetaphone1 ON
entities(phonetic_double_metaphone_1);

-- Fuzzy search (trigram)
CREATE INDEX idx_entities_name_trgm ON entities USING
gin(entity_name_normalized gin_trgm_ops);
CREATE INDEX idx_entities_aliases_trgm ON entities USING
gin(aliases_normalized gin_trgm_ops);
```

STAGE 2: Universal Intent Matcher (Queries New Table)

Step 2.1: Examine Current COTW Intent Matcher
Already done - we have this file

Identify what needs to change:
- Hardcoded table queries
- Missing phonetic matching
- Missing WHICH and IS question types

Step 2.2: Install Phonetic Matching Library
Code Terminal:
```
npm list metaphone
```

If not installed:
```
npm install metaphone
```
```
npm install double-metaphone
```

Step 2.3: Create Phonetic Code Generator Utility
File: backend/utils/phoneticGenerator.js
Purpose: Generate phonetic codes when entities are inserted

Step 2.4: Update Intent Matcher - Add Missing Question Types
Add to COTW:
- WHICH patterns
- IS/ARE patterns

Step 2.5: Update Intent Matcher - Replace Hardcoded Queries
Change from:

javascript
pool.query('SELECT character_name FROM character_profiles...')
To:

javascript
pool.query('SELECT entity_name FROM entities WHERE realm_hex_id = $1...', [realmId])

Step 2.6: Add Tiered Search Strategy

Tier 1: Exact match **Tier 2:** Phonetic match **Tier 3:** Fuzzy match (existing Fuse.js)

CRITICAL CHECKPOINTS

Before proceeding to each step, we must:

1. **Examine actual code** - not documentation
2. **Test in DB terminal** - verify queries work
3. **Verify with hex system** - not UUID
4. **Confirm isolation** - realm_hex_id filtering works
5. **No assumptions** - check everything

Questions Before We Start:

1. **Do you want to create the new entities table alongside existing tables first?** (Test without breaking current system)
2. **Should we start by examining your current database schema in detail?**
3. **Do you want to see your hex generator code first to understand how to use it properly?**
4. **Which step feels most important to tackle first?**

Let me know and we'll proceed one rigorous step at a time, examining real code and database state before making any changes.