Introduction

Project context and architectural rationale

The database used for this project was chosen for several reasons: it is part of a real, useful, and constantly evolving project, and it is situated within a project where I can put into practice what I am learning in all my degree courses.

I modeled this project's architecture on the human brain, utilizing two hemispheres (DatabaseIN and DatabaseOUT) with distinct but connected functions. DatabaseIN provides the infrastructure for an AI specialized in Italian and Finnish linguistic variants, dialects, and registers.

The necessity of a relational system became evident after a failed prototype during the winter semester. Using rigid Python scripts and flat files (CSV/JSON) via Colab proved insufficient to handle linguistic complexity. Critical semantic markers — such as punctuation for intent (?, !), synonyms, and tone variations — were lost as "data noise." This project solves that problem by using structured relational attributes to preserve the register and context of the language, which are essential for training a specialized AI.

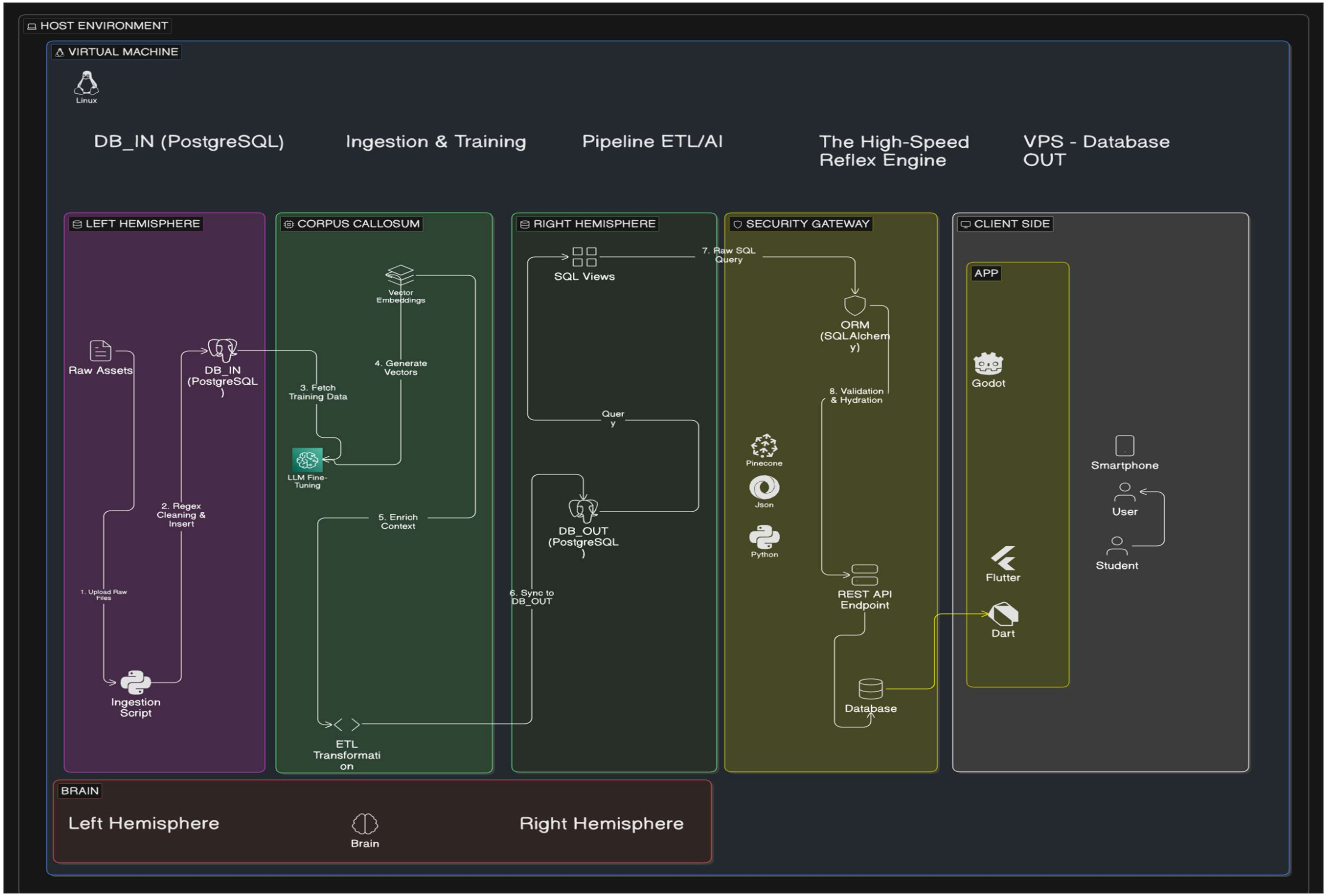
The efficiency of this pipeline was proven during the development of my app, COME NO. By integrating AI into the ingestion code for automated noise cleanup and tone recognition, I imported and categorized 9,000 words and 3,000 phrases in a few hours. This success validates, in my opinion, the "Double-Database" architecture I am now further developing.

Technical Architecture and Pipeline

The project follows a "separation of concerns" strategy:

- **Database IN (The Deep Knowledge Core):** The focus of this assignment. It acts as the source of truth for semantic training. It is populated using specialized datasets like *Universal Dependencies v2 (FreeCommons)*. I applied strict normalization (based on Chapters 3, 4 and 24 of book *"Fundamentals of Database Systems"*) to model hierarchical variants, dialects and tone intensity.
- **The Training Pipeline:** Structured data from Database IN is used to fine-tune open-source AI models.

- **Database OUT (The Reflex Engine):** The specialized AI then populates this denormalized database, which is optimized for high-speed delivery to the mobile app (FlutterFlow) and serves as an active in-app tutor.

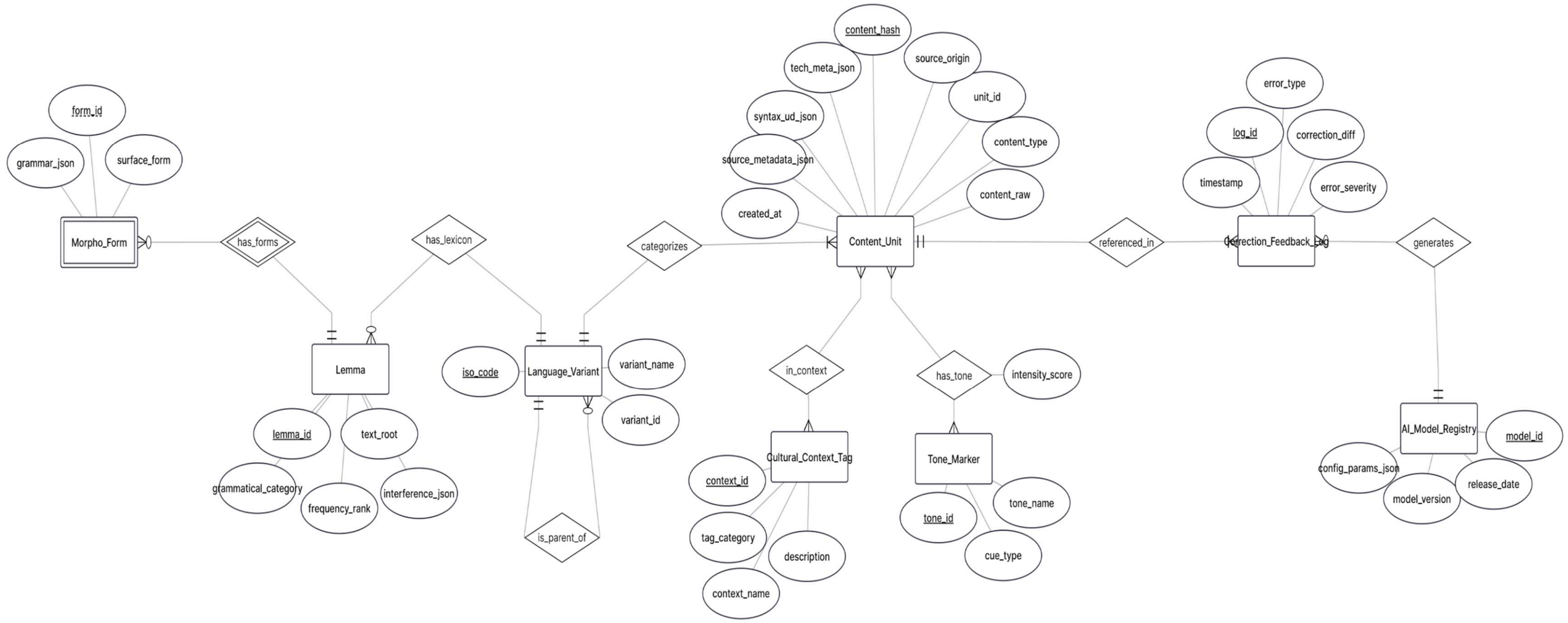


Assignment Focus and Compliance

For this course, I am implementing the **Database IN** using standard SQL DDL. This provides the structured foundation required for the AI training phase.

Entity Selection and Relational Core

- Linguistic lineages, including standard languages and regional dialects, are managed through a recursive relationship in the *Language_Variant* table to prevent data duplication.



- Principles of Second (2NF) and Third Normal Form (3NF) are upheld by the separation of *Lemma* and *Morpho_Form,* ensuring that grammatical categories are stored once and linked to specific surface forms only when necessary.
- Multi-dimensional tagging is facilitated by bridge tables (*Rel_ _ Content_Tone, Rel_Content Context*), through which unique technical weights — such as *intensity_score* and *relevance_score* — are assigned to each association.
- The implementation of JSON fields (e.g. *grammar_json syntax_ud_json, tech_meta_json*) is based on a decision to optimize data ingestion and processing.

Atomicity Preservation

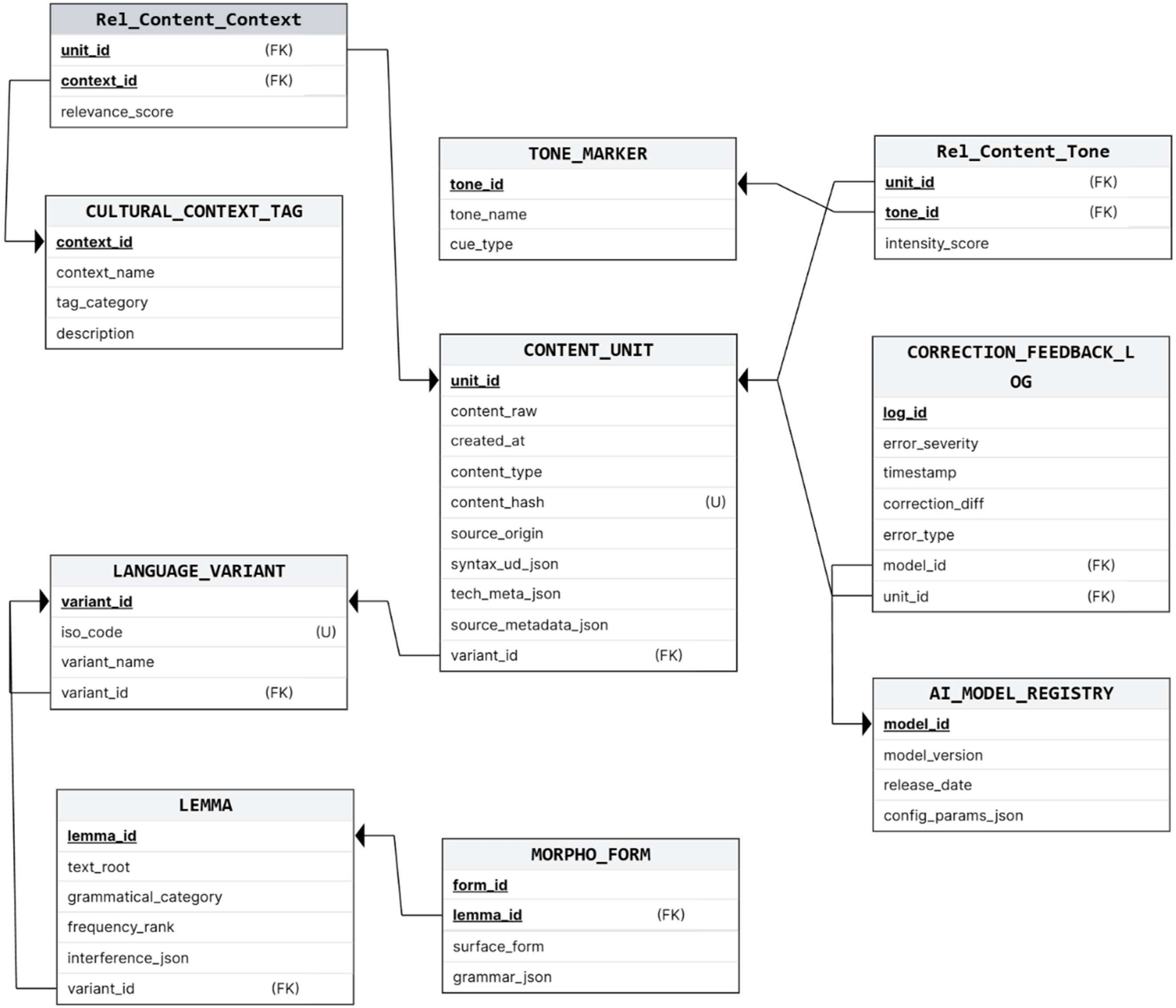
Standard flat data formats, such as CSV, are seamlessly integrated via basic SQL commands for simplified ingestion. Concurrently, complex linguistic objects, such as Universal Dependencies (UDv2) syntax trees, are treated as single atomic technical units within the JSON field to avoid massive `Join` use and improve AI data-loading efficiency.

Bypassing 1NF Constraints: structural context is preserved without information loss by treating the JSON object as a single unit, allowing for the storage of multi-dimensional linguistic data and acoustic parameters. A future-proof container is provided by the JSON/JSONB format for data and sound markers imported from sources like UDv2, accommodating varying metadata schemas without requiring full database migrations.

Integrity and Constraints - Composite Primary Keys

Referential integrity is ensured through the enforcement of Composite PKs (e.g`., form_id` + `lemma_id`) in junction tables and weak entities, preventing the processing of duplicate or orphan linguistic associations by the AI.

Audit trail is established via the *Correction_Feedback_Log*, whereby specific AI model versions from the *AI_Model_Registry* are traced directly to individual data corrections.


Rel_Content_Context
unit_id (FK)
context_id (FK)
relevance_score
CULTURAL_CONTEXT_TAG
context_id
context_name
tag_category
description
TONE_MARKER
tone_id
tone_name
cue_type
Rel_Content_Tone
unit_id (FK)
tone_id (FK)
intensity_score
CONTENT_UNIT
unit_id
content_raw
created_at
content_type
content_hash (U)
source_origin
syntax_ud_json
tech_meta_json
source_metadata_json
variant_id (FK)
CORRECTION_FEEDBACK_LOG
log_id
error_severity
timestamp
correction_diff
error_type
model_id (FK)
unit_id (FK)
LANGUAGE_VARIANT
variant_id
iso_code (U)
variant_name
variant_id (FK)
AI_MODEL_REGISTRY
model_id
model_version
release_date
config_params_json
LEMMA
lemma_id
text_root
grammatical_category
frequency_rank
interference_json
variant_id (FK)
MORPHO_FORM
form_id
lemma_id (FK)
surface_form
grammar_json