Data Dictionary

The data dictionary (see Tables 5 to 41) functions as a tool to properly yet concisely define or describe the crucial tables and their core attributes or fields in the Proficiency's utilized database schema. It outlines the intricate information and details of each table and its fields, along with the constraints and data types, information regarding their relationships with other tables through its primary keys and foreign keys, and other associated data elements associated with each table. Overall, the data dictionary is a proper reference for fully understanding the structure or schema and the organization of the system's database, facilitating effective data management and retrieval.

Table 5

DATA DICTIONARY OF UNIVERSITY_REGISTRATION_REQUESTS TABLE

Field Name	Constraints	Data Type	Description
university_registration_request_id	PK	int	Unique identifier for each university registration request
university_name	Not Null; Unique	varchar(255)	Full name of the university applying for registration.
acronym	Not Null; Unique	varchar(30)	Shortcode/acronym for the university.
street	Not Null	varchar(100)	The specific street name or road where the university or campus is located may also include building or unit numbers if applicable.
barangay	Null	varchar(100)	The administrative address component in the Philippines is required in the barangay, village, or subdivision where the university or campus is situated.
city	Not Null	varchar(50)	The city or municipality that governs the location of the university or campus.

province	Null	varchar(50)	The province in which the university or campus is located may be left blank for cities not under a province (e.g., Metro Manila cities).
postal_code	Not Null	varchar(10)	The university or campus address's official postal or ZIP code is stored as a string to allow leading zeros.
region	Null	varchar(30)	The administrative region as designated by the Philippine government (e.g., "Region VII," "NCR") is optional but useful for reporting and classification.
contact_university_email	Not Null, Unique	varchar(255)	Official contact email of the university for registration communication
contact_person_last_name	Not Null	varchar(100)	Last name of the university personnel initiating registration (future first admin).
contact_person_first_name	Not Null	varchar(100)	First name of the university personnel initiating registration (future first admin).
contact_person_middle_name	Null	varchar(100)	Middle name of the university personnel initiating registration (future first admin).
contact_person_school_id	Not Null	varchar(30)	Official university-issued ID number (for login/account association; must be unique within the university)

contact_person_position	Not Null	varchar(100)	Position/title of the registering personnel.
contact_person_email	Not Null, Unique	varchar(255)	Email of registering personnel (will be used as admin account).
contact_person_mobile_number	Not Null	varchar(20)	The mobile number of the registering personnel (may be used to contact them for further verification by the super admins if needed and highly necessary).
status	Not Null, Default: submitted	enum ('submitted,' 'in review,' 'approved,' 'rejected,' 'canceled')	Current status: submitted, reviewed, approved, rejected, etc.
submitted_at	Not Null, Default: Current Timestamp	timestamp	When the registration request was created/submitted.
final_decision_by	Null	int	ID of the super admin who made the final decision (approve/reject).
final_decision_at	Null	timestamp	Timestamp when the final decision was made.
updated_at	Not Null, Default: Current Timestamp On Update	timestamp	Last time the record was updated.

The University_Registration_Requests table (see Table 5) is a comprehensive log of all university registration attempts on the platform, recording successful submissions and unsuccessful or incomplete onboarding efforts. By retaining every attempt—regardless of the outcome—the system maintains a complete audit trail, enabling thorough tracking of user engagement and facilitating future review or follow-up actions on pending or abandoned requests. Importantly, an entry in the Universities table is only created when a registration request has been explicitly approved. At that point, the university is fully onboarded and activated in the system, and the first administrative user associated with the university is established and linked accordingly; meaning this table is intentionally made as a way to separate ongoing

university onboarding or registration from actual registered universities improving efficiency and performance.

Table 6

DATA DICTIONARY OF UNIVERSITY_REGISTRATION_ACTIONS TABLE

Field Name	Constraints	Data Type	Description
university_registration_action_id	PK	int	Unique identifier for each action / audit record.
university_registration_request_id	FK, Not Null; On Delete Cascade	int	References UNIVERSITY_REGISTRATION_REQUESTS.registration_request_id.
super_admin_id	FK, Null	int	ID of the super admin performing the action (NULL for system actions).
action_type	Not Null	enum('assigned','released','comment','approved','rejected','auto_released')	Type of action taken on the request. (assigned = taken for review / lock start).
action_note	Null	text	Optional free text note or reason (useful for approve/reject comments or handover notes).
started_at	Null	timestamp	When an assignment started. Set when action_type = 'assigned'.
last_activity_at	Not Null, Default: Current Time	timestamp	Last activity time for the action row (used to detect idle/abandoned assignments).

	stamp		
ended_at	Null	timestamp	When this action/assignment ended (set for released, approved, rejected, auto_released).
is_active	Not Null, Default: False	boolean	True for the current active assignment row; only one is_active = TRUE row should exist per request.

This append-only table (refer to Table 6) records every action on a university registration request. It exists to provide a clear audit trail and enforce accountability by showing who acted, what they did, and when. Each row logs the actor (or NULL for system), the action type (assigned, released, comment, approved, rejected, auto_released, etc.), key timestamps, and an optional note. The single row with is_active = TRUE (if any) indicates the current reviewer lock, while all other rows form the immutable history used for compliance, dispute resolution, and administrative reporting.

Table 7

DATA DICTIONARY OF UNIVERSITY_REGISTRATION_DOCUMENTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
university_registration_document_id	PK	int	Unique identifier for each university registration document.
university_registration_request_id	FK; Not Null	int	References UNIVERSITY_REGISTRATION_REQUESTS.registration_request_id.
document_type	Not Null	enum('accreditation_proof','school_license','supporting_letter','other')	Type/category of the document.
file_path	Not Null	varchar(255)	Storage path/URL.
original_file_name	Null	varchar(255)	Client filename

content_type	Null	varchar(100)	Specific file/content type of uploaded document (e.g., image/png, pdf, etc.)
file_size_bytes	Null	bigint unsigned	Size of the document in bytes.
uploaded_at	Not Null; Default: Current Timestamp	timestamp	When the document was uploaded.

This table (see table 7) holds the evidentiary files universities submit during onboarding so reviewers can verify institutional legitimacy against a concrete, dated record. Treating each upload as its own immutable entry preserves a reviewable history of what was provided and when, which is essential for auditability, compliance checks, and resolving disputes about missing or outdated paperwork. Categorized document types make checklist-driven reviews possible ("what's still missing?") and allow the UI to surface gaps quickly; stored file metadata enables safe retrieval, scanning, and storage governance (e.g., retention rules or virus checks). In practice, reviewers load all documents for the in-flight request, confirm completeness/validity, and proceed to a decision; operations teams can report on submission completeness and turnaround times without touching mutable sources.

Table 8

DATA DICTIONARY OF UNIVERSITIES TABLE

Field Name	**Constraints**	**Data Type**	**Description**
university_id	PK	int	Unique identifier for the university
registration_request_id	FK, Null; On Delete Set Null	int	References the originating University_Registration_Requests.registration_request_id upon approval.
university_name	Not Null; Unique	varchar (255)	Official full name of the university
acronym	Null; Unique	varchar (30)	Acronym or short version (e.g., UCLM); optional
street	Null	varchar (100)	The specific street name or road where the university or campus is located may also include building or unit numbers if applicable.

barangay	Null	varchar (100)	The administrative address component in the Philippines is required in the barangay, village, or subdivision where the university or campus is situated.
city	Not Null	varchar (50)	The city or municipality that governs the location of the university or campus.
province	Null	varchar (50)	The province in which the university or campus is located may be left blank for cities not under a province (e.g., Metro Manila cities).
postal_code	Not Null	varchar (10)	The university or campus address's official postal or ZIP code is stored as a string to allow leading zeros.
region	Null	varchar (30)	The administrative region as designated by the Philippine government (e.g., "Region VII," "NCR") is optional but useful for reporting and classification.
contact_university_email	Not Null	varchar (255)	Registrar/official contact of the university.
status	Not Null; Default: active	enum ('active,' 'inactive,' 'closed,' 'suspended')	Current operational status of the university
logo_path	Null	varchar(255)	Logo image path.
accreditation_type	Null	varchar(100)	Accreditation level of a university or school
created_at	Not Null, Default: Current Timestamp	timestamp	Record created timestamp of when it was approved
updated_at	Not Null, Default: Current Timestamp On Update	timestamp	Last updated timestamp; if modifications occur.

The Universities table (see Table 8) contains the authoritative records for all schools and institutions that have completed the registration and onboarding process within the Proficiency system. This table is the primary reference point for all university-specific data across the platform, ensuring consistency and integrity in data management. It is populated exclusively when a university's registration request has been reviewed and formally approved, signaling that the institution is fully onboarded and active in the system.

The Universities table is crucial in enforcing data isolation between tenants, supporting institution-specific branding, and enabling key SaaS operations and management aspects.

Table 9

DATA DICTIONARY OF DEPARTMENTS TABLE

Field Name	Constraints	Data Type	Description
department_id	PK	int	Unique identifier for each department.
university_id	FK, Not Null; On Delete Restrict	int	University this department belongs to.
parent_department_id	FK; Self-referencing department_id, Null	int	A self-referencing foreign key to identify or allow a department that belongs to a higher or major department. Null for top-level departments.
department_head_id	FK, Null; On Delete Set Null	int	A department head oversees the major department. References the user id (to support the possibility of non-faculty department heads)
department_name	Not Null	varchar(255)	Name of the major department (e.g., "Basic Education Department," "Senior High School Department," "College Department").
acronym	Null	varchar(30)	Optional acronym or code for the full name of a major department (e.g., "BED," "COL").
description	Null	text	Optional details.
created_at	Not Null, Default: Current Timestamp	timestamp	Timestamp of when the major department was created or added

updated_at	Not Null, Default: Current Timestamp On Update	timestamp	Timestamp of last updates to the major department
is_active	Not Null, Default: True	boolean	Responsible for keeping track of whether the major department is currently active or not; utilized for soft deletion/archiving.

Table 9.1
SPECIAL TABLE CONSTRAINTS OF DEPARTMENTS TABLE

Constraint Name	Type	Columns Involved	Description
chk_department_reference	CHECK CONSTRAINT	parent_department_id != department_id	This constraint is to ensure a department cannot reference itself or create a cycle.

The departments table (see Table 9 and 9.1) is the central repository for all academic departments within a university, capturing both top-level or major departments and sub-departments through a self-referencing structure. Additionally, it stores critical details such as unique department identifiers, university affiliations, department head assignments (including non-faculty heads), names, optional acronyms, and descriptions, while tracking creation, updates, and active status with timestamps and a boolean flag. This table ensures hierarchical organization, supports evaluation assignments by linking faculty to departments, and maintains data integrity across the platform, enabling efficient management of academic units and their associated resources.

Table 10
DATA DICTIONARY OF SCHOOL_TERMS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
school_term_id	PK	int	Unique identifier for each school year and semester with its corresponding modality.
university_id	FK, Not Null	int	University this school term belongs to specifically.
start_school_year	Not Null	int	Academic starting year.

end_school_year	Not Null	int	Academic end of school year.
semester	Not Null	enum(‘1st semester’, ‘2nd semester’, ‘Summer’)	Refers to the available semesters per school year (1st, 2nd semester, and summer).
teaching_modality	Not Null	enum('online,' 'modular,' 'hybrid,' 'face-to-face')	The specific teaching method or modality utilized for a school’s term (e.g., online, hybrid, face-to-face).
start_date	Null	date	This is optional but specifies when the term begins.
end_date	Null	date	This is optional but specifies when the term ends.
created_at	Not Null, Default: Current Timestamp	timestamp	The Created timestamp of the term or when the term was created in the database.
updated_at	Not Null, Default: Current Timestamp	timestamp	Last modified or updated timestamp.
is_active	Not Null, Default: True	boolean	For archiving but also to identify if the school term is currently active or not.

Table 10.1
SPECIAL TABLE CONSTRAINTS OF SCHOOL_TERMS TABLE

Constraint Name	Type	Columns Involved	Description

unique_university_term	UNIQUE CONSTRAINT	university_id, start_school_year, end_school_year, semester	This ensures that each university's combination of the school year and the semester is unique, preventing duplicate or overlapping academic terms.
chk_dates	CHECK CONSTRAINT	end_date > start_date	A constraint that ensures that the start and end dates of a school year (optional fields) are proper.

The School_Terms table (refer to Tables 10 and 10.1) defines each university's academic periods by specifying key attributes such as the school year, semester, and teaching modality (e.g., in-person, online, or hybrid). Every subject offering, student enrollment, and academic evaluation is associated with a specific term, making this table a foundational component for organizing and managing academic data. The School_Terms table enables robust analytics, precise scheduling, and flexible data filtering according to the platform's academic period or delivery mode by establishing clear term boundaries and modalities

Table 11

DATA DICTIONARY OF USER_REGISTRATION_TOKENS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
registration_token_id	PK	int	Unique identifier for each registration token record.
university_id	FK; Not Null	int	The specific university the registration token record is affiliated with; for the purpose of tenancy.
token_hash	Not Null	char(64)	The specific SHA-256 of the raw token; raw is never stored and is hashed.
intended_role	Not Null	enum('student','faculty','department_head','admin')	Intended role associated with the registration/invite token.

department_id	Null	int	Optional scope for students/faculty/department head tokens.
program_id	Null	int	Optional scope for student tokens.
max_uses	Not Null; Default: 1	int	The maximum allowed uses of the token; allows cohort tokens (large values) or single-use.
used_count	Not Null; Default 0	int	The amount of times the token was used (increments automatically on consumption)
expires_at	Null	timestamp	Optional expiry (e.g., end of enrollment week)
created_by_user_id	Not Null	int	The specific admin who created the token.
label	Null	varchar(100)	Optional human label of the token (e.g., “AY25 S1 Students”)
created_at	Not Null; Default: Current Timestamp	timestamp	When the token was created; for auditing.
revoked_at	Null	timestamp	When the token was softly revoked (invalidates token)

This table (see table 11 above) enables controlled, policy-driven self-service onboarding through scoped invite or cohort tokens, while preserving tenant boundaries. A token encodes enrollment intent (who may join, in what capacity, and optionally where in the org), plus guardrails like usage caps, expiration, and revocation so admins can run time-bound enrollment waves or one-off invites safely. Security is strengthened by storing only a hash of the raw token rather than the token itself, mitigating the blast radius of a data leak. Operationally, incoming links are validated against these constraints before any onboarding flow is unlocked; administratively, labels and creation metadata support audit trails (“who created this cohort and when?”) and post-mortems on enrollment issues. Analytically, usage counters

support capacity planning for peak waves and help identify abandoned or abused invitations that should be revoked.

Table 12

DATA DICTIONARY OF USER_REGISTRATION_REQUESTS TABLE

Field Name	Constraints	Data Type	Description
user_registration_request_id	PK	int	Unique identifier for each user registration request
university_id	FK; Not Null	int	University in which the user claims to be a part with/want to register with.
intended_role	Not Null	enium('student','faculty','department_head','admin')	The intended role of the user (or enforced by token)
department_id	FK; Null	int	Optional; but the specific department that the user may belong to if specified.
program_id	FK; Null	int	Optional; mainly for students to the current program they are associated with.
school_id	Not Null	varchar(30)	The user's claimed school id provided by their university.
email	Not Null	varchar(255)	Contact/login email.
last_name	Not Null	varchar(100)	The last name of the user registering.

first_name	Not Null	varchar(100)	The first name of the user registering.
middle_name	Null	varchar(100)	The middle name of the user registering.
registration_token_id	FK; Not Null	int	References the user_registration_tokens.registration_token_id. Wherein if this is available and valid; enables auto-approval.
status	Not Null; Default: submitted	enum('submitted','in_review','approved','rejected','canceled')	Status of the user registration
reviewed_by_user_id	FK; Null	int	The admin who reviewed the registration request.
reviewed_at	Null	timestamp	Specific timestamp of when the review occurred.
decision_note	Null	text	Optional note for providing reasons for either rejection or approval.
approved_user_id	FK; Null	int	The admin who approved the registration request.
created_at	Not Null; Default: Current Timestamp	timestamp	When the user registration was created and submitted
updated_at	Null; Default: Current Timestamp On Update	timestamp	When the registration was updated; as a way to audit.
is_active	Not Null; Default: True	boolean	To enforce one active request per identity/email per university or tenant.

This table (refer to table 12) is the intake ledger for people requesting access to a university tenant and records the full decision lifecycle for each request. It captures the applicant's claims and any token-driven policy shortcuts (including auto-approval when permitted), then moves through review states until a final decision is reached. The design supports accountability and defensibility: review and approval actors, timestamps, and decision notes provide clear provenance for why a request was approved or rejected. Operationally, timestamps enable SLA tracking, stale-item cleanup, and workload balancing; a guard prevents duplicate or competing in-flight requests for the same identity inside a tenant, reducing confusion and race conditions. On approval, downstream provisioning proceeds with high confidence because this record preserves "who/why/when" for the outcome; on rejection, the persisted rationale supports appeals and pattern analysis (e.g., common failure reasons that training or form changes could address).

Table 13

DATA DICTIONARY OF USER_REGISTRATION_DOCUMENTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
user_registration_document_id	PK	int	Unique identifier for each user registration document.
user_registration_request_id	FK; Not Null	int	References USER_REGISTRATION_REQUESTS.user_registration_request_id.
document_type	Not Null	enum('accreditation_proof','school_license','supporting_letter','other')	Type/category of the document.
file_path	Not Null	varchar(255)	Storage path/URL.
original_file_name	Null	varchar(255)	Client filename

content_type	Null	varchar(100)	Specific file/content type of uploaded document (e.g., image/png, pdf, etc.)
file_size_bytes	Null	bigint unsigned	Size of the document in bytes.
uploaded_at	Not Null; Default: Current Timestamp	timestamp	When the document was uploaded.

The table above (see table 13) stores the evidentiary files individual users submit during onboarding (e.g., IDs, permits, supporting letters) so reviewers can verify identity and affiliation against a durable, time-stamped record. Treating each upload as its own immutable entry preserves an auditable trail of "what was provided, and when," which simplifies dispute resolution and compliance checks. Categorized document types enable checklist-style reviews and quick detection of missing prerequisites, while stored file metadata supports safe retrieval, scanning, and storage governance (retention, size limits, MIME validation). In practice, the reviewer UI loads all documents for a pending request, allowing rapid completeness/validity checks; operations can then report on submission completeness and turnaround without touching mutable sources.

Table 14

DATA DICTIONARY OF USERS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
user_id	PK	int	Unique identifier for each user
university_id	FK; Not Null	int	A foreign key to the universities table's university_id.
school_id	Not Null	varchar(30)	Official university-issued ID number (for login/account association; must be unique within the university)

username	Not Null	varchar(50)	Login username; formatted initially but editable by the user.
email	Not Null, Unique	varchar(255)	Unique email for password reset and contact; must be unique system-wide
password_hash	Not Null	varchar(255)	The user's password is hashed for secure authentication.
admin_pin_code_hash	Null	varchar(255)	If the user is an admin, for added security they have a six (6) digit pin code that they need to enter to access their account.
last_name	Not Null	varchar(100)	A user’s last name
first_name	Not Null	varchar(100)	A user’s first name
middle_name	Null	varchar(100)	A user's middle name is optional
profile_photo_path	Null	varchar(255)	Path or URL to the user’s profile photo
status	Not Null, Default: Active	enum ('active,' 'inactive,' 'archived,' 'locked,' 'suspended')	Current status of the account.
email_verified	Not Null, Default: False	boolean	Whether the user’s email is verified
last_login_at	Null	timestamp	Last login timestamp

created_at	Not Null, Default: Current Timestamp	timestamp	Record creation date.
updated_at	Not Null, Default: Current Timestamp	timestamp	Last update date.

Table 14.1
SPECIAL TABLE CONSTRAINTS OF USERS TABLE

Constraint Name	Type	Columns Involved	Description
unique_user_university_school_id	UNIQUE CONSTRAINT	university_id, school_id	Prevents duplicate assignment of student/employee numbers in the same institution which could cause login confusion.
unique_university_username	UNIQUE CONSTRAINT	university_id, username	Allows same usernames across different universities but unique usernames or no duplicate usernames within a university (e.g., username1 in uclm and username1 in ucb but no duplicate username1 on uclm).

The Users table (shown in Table 14 and 14.1 above) is the centralized repository for everyone interacting with the Proficiency system, except for super administrators. This table stores essential information for each user, including their profile details, account credentials, assigned role, and relevant audit data. Structured to ensure one unique user per row, it supports various user types—students, instructors, and administrators—by leveraging a flexible role field. In addition to basic profile data, the table records account status (such as active or suspended), tracks login activity, and can store an optional profile photo. All associations to academic activities, evaluations, and subject enrollments reference users through this table, making it a foundational element for user management and platform security.

Table 15
DATA DICTIONARY OF USER_ROLES TABLE

Field Name	**Constraints**	**Data Type**	**Description**
user_role_id	PK	int	Unique identifier for each user-role assignment.
user_id	FK, Not Null	int	References Users.user_id.

role	Not Null	enum('student,' 'faculty,' 'admin,' 'department_head')	The role was assigned to the user for this record.
assigned_at	Not Null, Default Current Timestamp	timestamp	Timestamp when the role was assigned.
revoked_at	Null	timestamp	Timestamp when the role was revoked (null if still active).
is_active	Not Null, Default True	boolean	Indicates if the user's role is currently active.

Table 15.1
SPECIAL TABLE CONSTRAINTS OF USER_ROLES TABLE

Constraint Name	Type	Columns Involved	Description
unique_user_id_role_is_active	UNIQUE CONSTRAINT	user_id, role, is_active	To enforce that a user cannot have two active records of the same role.

This junction table (refer to Table 15 and 15.1) enables support for multiple concurrent roles per user. It tracks the lifecycle of each role assignment with start and optional end dates, allowing flexible role management and historical auditing. This design decouples role data from the Users table, enhancing extensibility and maintaining data integrity.

Table 16
DATA DICTIONARY OF SUPER_ADMINS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
super_admin_id	PK	int	Unique identifier for each super admin.
username	Not Null, Unique	varchar(50)	Super admin login username.
email	Not Null, Unique	varchar(255)	Email (must be unique globally).

password_hash	Not Null	varchar(255)	The following is the super admin password, which has been hashed for secure authentication.
super_admin_pin_code_hash	Not Null	varchar(255)	Six (6) digit PIN code for further security. Ensuring access to super admin accounts have multi-factor authentication.
last_name	Not Null	varchar(100)	The surname of the super admin
first_name	Not Null	varchar(100)	First name of the super admin
middle_name	Null	varchar(100)	The middle name of the super admin is optional
profile_photo_path	Null	varchar(255)	Optional profile photo path
status	Not Null, Default: Active	enum ('active,' 'inactive,' 'locked')	An account's status
last_login	Null	timestamp	Last login timestamp
created_at	Not Null, Default: Current Timestamp	timestamp	Created timestamp of account
updated_at	Not Null, Default: Current Timestamp	timestamp	Last updated timestamp

This table (Table 16) stores information for system-wide super admin users, who oversee platform-level operations, user onboarding, and audit logs across the entire system. Super admins are distinct from

university-specific users to ensure enhanced security and strict multi-tenancy isolation. The table maintains essential authentication credentials, account status indicators, and contact details for each super admin, enabling secure access and efficient communication at the platform level.

Table 17

DATA DICTIONARY OF USER_DEPARTMENT_ASSIGNMENTS TABLE

Field Name	Constraints	Data Type	Description
user_department_id	PK	int	Unique identifier for each user-department assignment record.
user_id	FK; Not Null; On Delete CASCADE	int	References the user assigned to the department.
department_id	FK; Not Null; On Delete CASCADE	int	References the department the user is affiliated with.
program_id	FK; Null	int	(Optional) References the academic program/course if applicable.
assigned_at	Not Null	timestamp	Timestamp when the assignment began.
revoked_at	Null	timestamp	Timestamp when the assignment ended (null if still active).
is_active	Not Null; Default True	boolean	Indicates if the assignment is currently active.

This table (Table 17) maps users to departments and, optionally, programs supporting many-to-many and historical affiliations. It enables precise role-based access control, department-level analytics, and tracking of changes in departmental assignments over time for faculty, students, admins, and department heads.

Table 18

DATA DICTIONARY OF PROGRAMS TABLE

Field Name	Constraints	Data Type	Description
program_id	PK	int	Unique identifier for each program/course/track/grade level.
university_id	FK; Not Null	int	The specific university where the program belongs to.

department_id	FK, Not Null; On Delete Restrict	int	References Departments.department_id. Helps track on what department essentially owns the department
program_code	Not Null, Unique	varchar(30)	University's code for the program (e.g., "BSIT," "STEM," “G1,” etc.).
program_name	Not Null	varchar(255)	Official program/course/track/grade name.
program_level	Null	enum(‘basic education’, 'senior_high','college','graduate')	The specific educational stage or level of the program (e.g., basic education, senior high, college, or graduate level).
description	Null	text	Additional notes/details.
created_at	Not Null, Default: Current Timestamp	timestamp	Record creation date.
updated_at	Not Null, Default: Current Timestamp	timestamp	Last modified date.
is_active	Not Null, Default: True	boolean	Active/archive flag.

This table (Table 18) catalogs all academic programs within the university, including programs, courses, tracks, and grade levels. Each entry is associated with a specific department, allowing for clear organizational structure and relationships. The table accommodates various educational programs, such as college degree programs, senior and junior high school tracks, and basic education levels. This comprehensive structure enables efficient filtering, robust reporting, and seamless linkage with curriculum data for academic planning and management.

Table 19
DATA DICTIONARY OF SUBJECTS TABLE

Field Name	Constraints	Data Type	Description
subject_id	PK	int	Unique identifier for the subject record.
university_id	FK; Not Null; On Delete Restrict	int	The university that owns this subject definition.

subject_code	Not Null	varchar(30)	Unique code per subject (e.g., “MATH101”).
name	Not Null	varchar(255)	Subject title (e.g., “Calculus 1”).
description	Null	text	Optional longer description.
units	Null	int	Number of units/credits (if applicable).
is_active	Not Null, Default: Active	boolean	Active/archive flag.
created_at	Not Null, Default: Current Timestamp	timestamp	Record creation.
updated_at	Not Null, Default: Current Timestamp	timestamp	Last updated.

Table 19.1
SPECIAL TABLE CONSTRAINTS OF SUBJECTS TABLE

Constraint Name	Type	Columns Involved	Description

unique_subjects_subject_code	UNIQUE CONSTRAINT	university_id, subject_code	This constraint ensures that each subject code remains unique within each university, preventing duplicate subject codes for the same university while allowing different universities to use the same code if desired.

This table (see Table 19 above; for its unique constraint see Table 19.1) stores comprehensive definitions of all subjects offered by the university. It includes subject codes, official names, unit values, and current status, providing a centralized reference for curriculum management. The table supports key functions such as tracking subjects within academic programs, facilitating the creation of subject offerings, and enabling historical analysis for curriculum evaluation and planning.

Table 20
DATA DICTIONARY OF PROGRAM_SUBJECTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
program_subject_id	PK	int	Unique identifier for the link of a program to subjects
program_id	FK; Not Null; On Delete Cascade	int	References the associated program.
subject_id	FK; Not Null; On Delete Cascade	int	References the associated subject.

Table 20.1
SPECIAL TABLE CONSTRAINTS OF PROGRAM_SUBJECTS TABLE

Constraint Name	Type	Columns Involved	Description
unique_program_subject_assignment	UNIQUE CONSTRAINT	program_id, subject_id	Ensures no duplicate program-subject associations.

This junction table (as shown in Tables 20 and 20.1) manages the many-to-many relationship between academic programs and subjects, enabling each subject to be associated with multiple programs and each program to include various subjects. This flexible structure supports dynamic curriculum design across different tracks and grade levels, allowing universities to efficiently map subjects to diverse academic

pathways and ensure consistency in curriculum offerings. Lastly, it implements a unique constraint with "(program_id, subject_id)" to ensure no duplicate program-subject associations occur.

Table 21
DATA DICTIONARY OF SUBJECT_DEPARTMENTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
subject_department_id	PK	int	Unique identifier for the link between subjects and departments
subject_id	FK; Not Null; On Delete Cascade	int	References the associated subject.
department_id	FK, Not Null; On Delete Cascade	int	References the department associated with the subject or the department that owns or has affiliations with the subject.

Table 21.1
SPECIAL TABLE CONSTRAINTS OF SUBJECT_DEPARTMENTS TABLE

Constraint Name	Type	Columns Involved	Description
unique_subject_department_assignment	UNIQUE CONSTRAINT	subject_id, department_id	Prevents duplicate subject-department mappings.

This junction table (refer to Tables 21 and Table 21.1) defines the many-to-many associations between subjects and departments, allowing each subject to be linked to multiple departments and vice versa. This structure supports curriculum flexibility and enables shared or general subjects—such as Mathematics or Physical Education—to be included in the offerings of various academic units. It also enhances analytics and reporting capabilities by accurately reflecting cross-departmental subject usage and collaboration. This table also has a unique constraint with "(subject_id, department_id)" to prevent duplicate subject-department mappings.

Table 22
DATA DICTIONARY OF SUBJECT_OFFERINGS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
subject_offering_id	PK	int	Unique identifier for the subject offering.
subject_offering_code	Not Null	varchar(30)	University-specific code for this offering (e.g., EDP code).
university_id	FK, Not Null	int	References the owning university for tenancy enforcement.

subject_id	FK; Not Null; On Delete Cascade	int	References the subject being offered.
school_term_id	FK; Not Null; On Delete Cascade	int	References the term/semester of this offering.
faculty_user_id	FK; Null; On Delete Set Null	int	References the faculty assigned to teach this offering.
department_id	FK, Not Null; On Delete Cascade	int	References the department the subject offering is associated with..
program_id	FK	int	(Optional) Links this offering to a program if relevant for analytics.
start_time	Not Null	time	Start time of the subject offering
end_time	Not Null	time	End time of the subject offering
days	Not Null	varchar(15)	The subject's schedule is offered specifically on days (e.g., MWF, TTH, etc.).
room	Null	varchar(50)	Room assignment (as plain text).
is_active	Not Null, Default: True	boolean	Active/archive flag.
created_at	Not Null, Default: Current Timestamp	timestamp	Creation timestamp.
updated_at	Not Null, Default: Current Timestamp	timestamp	Last update.

Table 22.1
SPECIAL TABLE CONSTRAINTS OF SUBJECT_OFFERINGS TABLE

Constraint Name	Type	Columns Involved	Description

unique_subject_offerings	UNIQUE CONSTRAINT	subject_offering_code, university_id	Ensures uniqueness of offering codes per university.

This table (see Table 22 and Table 22.1) defines each unique subject offering offered during an academic term, capturing key details such as the specific subject, scheduled time, assigned faculty member, department, and section designation. Each subject or class offering is identified by a unique offering code (EDP), which ensures distinctness across terms and facilitates efficient schedule management. This structure supports accurate student enrollment, streamlined class assignments, and precise faculty evaluation processes, enabling clear organization and reporting within the academic system. It also has a special constraint to ensure the uniqueness of offering codes per university through "(subject_offering_code, university_id)."

Table 23
DATA DICTIONARY OF STUDENT_ENROLLMENTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
enrollment_id	PK	int	Unique identifier for the enrollment record.
student_user_id	FK; Not Null; On Delete CASCADE	int	References the student (users.user_id with role='student').
subject_offering_id	FK; Not Null; On Delete CASCADE	int	References the subject offering enrolled in.
enrolled_at	Not Null	timestamp	When the enrollment was made.
is_active	Not Null, Default: True	boolean	Active/archive flag (for dropped/withdrawn cases).

Table 23.1
SPECIAL TABLE CONSTRAINTS OF STUDENT_ENROLLMENTS TABLE

Constraint Name	Type	Columns Involved	Description
unique_student_enrollments	UNIQUE CONSTRAINT	student_user_id, subject_offering_id	Prevents duplicate enrollments.

This table (Table 23 and 23.1) records student enrollments for specific subject offerings in each academic term, establishing a clear link between students and the classes they attend. By maintaining accurate enrollment data, the system ensures that only officially enrolled students can evaluate faculty for their respective classes. Additionally, this table underpins historical analytics, supports detailed per-term filtering, and facilitates comprehensive reporting on student participation and academic trends over time.

Table 24

DATA DICTIONARY OF SUBJECT_OFFERING_FACULTY_HISTORY TABLE

Field Name	Constraints	Data Type	Description
subject_offering_faculty_history_id	PK	int	Unique record.
subject_offering_id	FK	int	Reference to the subject offering
faculty_user_id	FK	int	The faculty assigned
assignment_start_date	Not Null	timestamp	Start datetime of assignment
assignment_end_date	Null	timestamp	End datetime of assignment (nullable if current)
changed_by_user_id	FK, Null	int	Admin who recorded the change (audit).
changed_reason	Null	varchar(255)	Short reason (e.g., leave, schedule conflict).
created_at	Not Null, Default: Current Timestamp	timestamp	Audit timestamp for the record.

The subject_offering_faculty_history table (as shown in Table 24 above) records the detailed history of faculty assignments for each subject offering, capturing the exact periods during which specific faculty members were responsible for teaching a particular class or section. This temporal tracking is essential for accurately associating evaluations and academic records with the correct instructor, especially in mid-term faculty replacements or reassignments. By maintaining a comprehensive log of assignment intervals, the table ensures data integrity and supports precise historical reporting, auditing, and analysis of faculty performance over time.

Table 25

DATA DICTIONARY OF EVALUATION_FORM_TEMPLATES TABLE

Field Name	Constraints	Data Type	Description
form_template_id	PK	int	Unique identifier for the evaluation form template.
university_id	FK	int	References to the university that owns this template.
name	Not Null	varchar(100)	Admin-defined name for the template/version (e.g., “Midterm 2024 v1”).
description	Null	text	(Optional) Extra details or notes.
likert_scale_template_id	FK; Default: 1	int	A foreign key referencing the likert_scale_template table to define the utilized likert scale for the questions (e.g., 4-scale, 5-scale, or 7-scale).
instructions	Null	text	These are custom instructions or legends to show at the start of the form.
status	Not Null, Default: Draft	enum ('draft,' 'active', 'assigned', 'archived')	Current lifecycle state.
created_by_user_id	Not Null	int	References users.user_id who created the template and primarily needs only to be an admin; they are the only ones with access to evaluation form management.

created_at	Not Null, Default: Current Timestamp	timestamp	Creation time.
updated_by	FK; Null	int	The admin who made updates or changes to the evaluation form template.
updated_at	Not Null, Default: Current Timestamp	timestamp	Last update time.

This table (see Table 25) stores all evaluation form templates and their versions for each university. Each form template can include custom instructions and support status tracking throughout its lifecycle. Templates and their versions are retained to support historical analytics, auditing requirements, and future reuse or reference.

Table 26

DATA DICTIONARY OF LIKERT_SCALE_TEMPLATES TABLE

Field Name	Constraints	Data Type	Description
likert_scale_template_id	PK	int	Unique identifier for the likert-scale template.
likert_name	Not Null; Unique	varchar(100)	A descriptive name for the scale (e.g., "Standard 1-5 Scale," "Detailed 1-7 Scale").
min_value	Not Null; Default: 1	tinyint	The minimum value of the scale (e.g., 1).
max_value	Not Null	tinyint	The maximum value of the scale (e.g., 5).

The table above (see table 26) defines the institution's reusable Likert scales so evaluation forms remain consistent and comparable across terms and cohorts. By centralizing named scales (e.g., "Standard 1–5," "Detailed 1–7") with explicit min/max bounds, forms can reference a canonical definition rather than inlining ad-hoc ranges—reducing schema drift, preventing scoring bugs, and enabling longitudinal analytics. Template indirection also allows safe evolution: new forms can adopt a revised scale without rewriting historical responses, and analytics can map results to a normalized frame when needed. Operationally, this table powers authoring (pick a scale by name), validation (reject responses outside bounds), and downstream scoring (consistent numeric semantics across all items).

Table 27

DATA DICTIONARY OF EVALUATION_CRITERIA TABLE

Field Name	Constraints	Data Type	Description
evaluation_criteria_id	PK	int	Unique identifier for the criterion.
form_template_id	FK; Not Null	int	References the parent evaluation form template.
criterion_name	Not Null	varchar(100)	Criterion or section title (e.g., “Teaching Methods”).
criterion_description	Null	text	Optional further explanation for evaluators.
weight	Not Null	decimal(5,2)	Percent weight for this criterion (sum must be 100 across criteria per evaluation form).
order	Not Null	int	Display order within the form.

Table 27.1
SPECIAL TABLE CONSTRAINTS OF EVALUATION_CRITERIA TABLE

Constraint Name	Type	Columns Involved	Description
unique_likert_answer	UNIQUE CONSTRAINT	form_template_id, order	Guarantees that within a single form template, each criterion’s order value is used only once.

This table (shown in Table 27 and 27.1 above) represents an individual section or criterion within a specific evaluation form template, with each entry assigned a weight to support aggregated scoring. It maintains the ordering of sections or criteria within the form and enables dynamic creation and editing of evaluation forms.

Table 28
DATA DICTIONARY OF EVALUATION_QUESTIONS TABLE

Field Name	Constraints	Data Type	Description
question_id	PK	int	Unique identifier for the question.
form_template_id	FK; Null	int	References the form template for open-ended questions. If the question_type is open-ended, this should not be null but criterion_id is null.

criterion_id	FK; Null	int	References the criterion (only for Likert-type questions). If the question_type is likert this should not be null but form_template_id is null.
question_text	Not Null	varchar(500)	The question prompt.
question_type	Not Null	enum('likert', 'open_ended')	The type of question.
order	Not Null	int	Ordering index of the question within its criterion or form.
is_required	Not Null, Default True	boolean	Whether this question is required to be answered.
is_active	Not Null, Default: True	boolean	Archive flag (for question edits/deletion).

Table 28.1
SPECIAL TABLE CONSTRAINTS OF EVALUATION_CRITERIA TABLE

Constraint Name	Type	Columns Involved	Description
unique_likert_answer	UNIQUE CONSTRAINT	criterion_id, order	This helps prevent duplicate question ordering under each criterion.

This table (Table 28 and 28.1) stores all Likert-scale (numerical) questions associated with each criterion for every version of an evaluation form. It maintains the order of questions within each criterion and ensures that responses are accurately mapped to the correct form version and question.

Table 29
DATA DICTIONARY OF EVALUATION_FORM_ASSIGNMENTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
form_assignment_id	PK	int	Unique identifier for this assignment record.
university_id	FK; Not Null	int	The university this assignment applies to.

evaluator_role	Not Null; Default: both	enum (‘student’, ‘department_head’, ‘both’)	This field specifies the evaluator type this form is being assigned to. The default for this is both, meaning the same evaluation form assigned for the two main evaluators.
school_term_id	FK; Not Null	int	The school term (semester) for this evaluation period.
assessment_period	Not Null	enum ('midterm,' 'finals')	Assessment period for this assignment.
form_template_id	FK; Not Null	int	The assigned form template/version for this period.
start_date_time	Not Null	timestamp	Starting date and time for the evaluation
end_date_time	Not Null	timestamp	End date and time for the evaluation
assigned_at	Not Null, Default: Current Timestamp	timestamp	When this assignment was made.
status	Not Null, Default: Active	enum ('active', 'archived', 'cancelling', 'cancelled')	Clarifies and confirms if this is an active assignment.
cancellation_reason	Null	varchar(255)	Stores the pre-defined or custom reason for cancellation
cancellation_notes	Null	text	Stores the internal administrative notes for “Other” cancellation reasons.
cancelled_by_user_id	FK; Null	int	References the admin who initiated the cancellation

cancelled_at	Null	timestamp	The timestamp when the cancellation was processed.

Table 29.1
SPECIAL TABLE CONSTRAINTS OF EVALUATION_FORM_ASSIGNMENTS TABLE

Constraint Name	Type	Columns Involved	Description
unique_student_enrollments	UNIQUE CONSTRAINT	university_id, school_term_id, assessment_period, evaluator_role	Ensures one form per term/evaluator type.
chk_date_time	CHECK CONSTRAINT	end_date_time > start_date_time	Simple validation check for assignment period.

This table (see Table 29 and 29.1 above) associates a specific evaluation form template with a university, academic term, and assessment period (such as midterm or finals). It enforces that only one active form exists per university, term, and assessment period, ensuring clarity and consistency. This table efficiently identifies and retrieves the 'current' evaluation form applicable for any given university and assessment window.

Table 30
DATA DICTIONARY OF EVALUATION_LIKERT_ANSWERS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
likert_answer_id	PK	int	Unique answer record.
submission_id	FK; Not Null; On Delete CASCADE	int	References the related evaluation submission.
question_id	FK; Not Null	int	This helps identify or associate to what particular likert question does the answer pertain to; the question being answered (likert-type).
answer_value	Not Null	tinyint	The numeric answer value (must fall within the linked scale’s range).
created_at	Not Null	timestamp	Timestamp of record creation.
updated_at	Not Null	timestamp	Timestamp of last update.

Table 30.1

SPECIAL TABLE CONSTRAINTS OF EVALUATION_LIKERT_ANSWERS TABLE

Constraint Name	Type	Columns Involved	Description
unique_likert_answer	UNIQUE CONSTRAINT	submission_id, question_id	To ensure only one answer per question per submission

This table (refer to Table 30 and 30.1 above) captures all numerical (Likert-scale) responses submitted through evaluation forms. Each response is linked to the student, subject offering, specific form version, criterion, and individual question, enabling comprehensive analytics, reporting, and creation of audits of evaluation data.

Table 31

DATA DICTIONARY OF EVALUATION_OPEN_ENDED_ANSWERS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
open_ended_answer_id	PK	int	Unique open-ended answer record.
submission_id	FK; Not Null; On Delete CASCADE	int	Links to Evaluation_Submissions
question_id	FK; Not Null	int	References the questions table and on which open-ended question.
open_ended_answer	Not Null	text	The student's written feedback.
created_at	Not Null	timestamp	Timestamp of record creation.
updated_at	Not Null	timestamp	Timestamp of last update.

Table 31.1

SPECIAL TABLE CONSTRAINTS OF EVALUATION_OPEN_ENDED_ANSWERS TABLE

Constraint Name	Type	Columns Involved	Description
unique_open_ended_answer	UNIQUE CONSTRAINT	submission_id, question_id	To ensure only one answer per question per submission

This table (see Table 31 and 31.1) stores all open-ended feedback responses, associating each entry with the corresponding form, specific question, student, and class. This structure supports future AI-driven analysis, comprehensive historical tracking, and review of qualitative evaluation data.

Table 32

DATA DICTIONARY OF EVALUATION_SUBMISSIONS TABLE

Field Name	Constraints	Data Type	Description
submission_id	PK	int	Unique evaluation session identifier.
evaluator_user_id	FK, Not Null	int	The student or department head who submitted the evaluation. References the user_id table with its linkage to user roles to ensure this is a student or a department head.
evaluator_role	Not Null	enum (‘student’, ‘department_head’)	Indicates the role of the evaluator (e.g., whether it is a student or a department head).
faculty_user_id	FK, Null	int	This field, faculty_user_id, refers to the users table’s user.user_id for the faculty being evaluated (only for department heads that are evaluating)
subject_offering_id	FK, Not Null	int	Subject offering evaluated.
form_assignment_id	FK, Not Null	int	Evaluation form version used.
university_id	FK, Not Null	int	Tenant university.
evaluation_start_time	Not Null	timestamp	Timestamp when evaluation started (for timing validation).

evaluation_end_time	Nullable	timestamp	Timestamp when the evaluation was completed/submitted.
status	Not Null, Default 'pending'	enum('pending,' 'submitted')	Submission status for audit and validation control.
created_at	Not Null	timestamp	Record creation timestamp.
updated_at	Not Null	timestamp	Last update timestamp.

Table 32.1
SPECIAL TABLE CONSTRAINTS OF EVALUATION_SUBMISSIONS TABLE

Constraint Name	Type	Columns Involved	Description
unique_evaluation_submissions	UNIQUE CONSTRAINT	subject_offering_id, evaluator_user_id, evaluator_role, form_assignment_id	Ensures no duplicate submissions.
chk_role_logic	CHECK CONSTRAINT	(evaluator_role = 'department_head' AND faculty_user_id IS NOT NULL) OR (evaluator_role = 'student' AND faculty_user_id IS NULL)	Helps enforce role logic.

This entity (refer to Table 32 and 32.1 above) represents a single evaluation session conducted by a student for a specific faculty member's subject offering, or a department head evaluating a faculty within their jurisdiction, utilizing the assigned evaluation form tailored for that course. It plays a crucial role in enforcing timing validation rules—such as ensuring that the evaluation takes at least 45 seconds to complete—to promote thoughtful responses. Additionally, it is essential for tracking the submission state, enabling the system to monitor evaluation timing and detect any attempts to submit incomplete or rushed feedback, thereby supporting the overall integrity and reliability of the faculty evaluation process.

Table 33
DATA DICTIONARY OF OPEN_ENDED_SENTIMENTS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
open_ended_answer_sentiment_id	PK	int	Unique record ID.

open_ended_answer_id	FK	int	The open-ended answers that are being analyzed.
model_name	Not Null, Default: xlm-roberta-fine-tuned	varchar(100)	The name of the model utilized for sentiment analysis.
model_version	Not Null	varchar(80)	The version of the utilized model for sentiment analysis
predicted_sentiment_label	Not Null	varchar(10)	The predicted sentiment label is provided by a sentiment analysis model/fusion.
predicted_sentiment_label_score	Not Null	decimal(5,4)	Score/probability of predicted sentiment or otherwise known the highest sentiment label (0.0000–1.0000).
positive_score	Not Null	decimal(5,4)	Score/probability of positive sentiment (0.0000–1.0000).
neutral_score	Not Null	decimal(5,4)	Score/probability of neutral sentiment (0.0000–1.0000).
negative_score	Not Null	decimal(5,4)	Score/probability of negative sentiment (0.0000–1.0000).
accuracy	Null	decimal(5,4)	Accuracy of this model's prediction (nullable if unavailable).

confidence	Null	decimal(5,4)	Confidence score (may differ from highest_sentiment_score).
created_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	When a record was created.
updated_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Last update timestamp.

This table (refer to Table 33 above) is crucial for storing all individual sentiment analysis outputs from multiple AI models on every open-ended feedback. It enables your system to maintain comprehensive historical records for model comparison, detailed sentiment breakdowns, and fusion of primary models' outputs. This supports transparency and auditability, allowing different roles to view filtered or combined sentiment results as specified. Storing fine-grained sentiment scores and predicted labels facilitates rich visualizations (e.g., sentiment distributions, gauges) and analytical reporting.

Table 34

DATA DICTIONARY OF OPEN_ENDED_KEYWORDS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
keyword_id	PK	int	Unique keyword record ID.
university_id	FK	int	Adds explicit tenant scoping for isolation and filtering.
open_ended_answer_id	FK	int	Related open-ended answer or the open-ended answer where the keyword was derived.
keyword	Not Null	varchar(255)	Extracted keyword/phrase from KeyBERT.
relevance_score	Not Null	decimal(5,4)	Relevance/confidence score (0.0000–1.0000).

rank_order	Null	int	1 = top phrase, etc. Helpful for "Top-N" UI.
ngram_size	NULL	int	This basically records the total number of words that was considered a keyword (e.g., 1=unigram, 2=bigram, etc).
extractor_version	Not Null	varchar(50)	The specific version of the model utilized for keyword extraction.
sentiment_label	Not Null	varchar(10)	Sentiment category of the keyword (from context or aggregation).
created_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	When the keyword was stored.
updated_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Last update timestamp.

This table (see Table 34) stores detailed keywords or key phrases extracted from each open-ended answer using KeyBERT, categorized by sentiment. It supports visualizations like sentiment-specific word clouds and thematic analysis at various aggregation levels. Accurate relevance scoring allows filtering to top keywords, maintaining visualization clarity. This is critical for producing actionable insights from qualitative feedback, helping faculty and administrators identify common themes and areas for improvement.

Table 35

DATA DICTIONARY OF NUMERICAL_AGGREGATES TABLE

Field Name	**Constraints**	**Data Type**	**Description**
university_id	FK, Not Null	int	University that owns the record; enables multi-tenant separation.
school_term_id	FK, Not Null	int	The school year + semester this result belongs to.
assessment_period	Not Null	enum('midterms','finals')	Which portion of the term the result covers.

subject_offering_id	FK, Not Null	int	The exact class/section evaluated (the teaching assignment).
subject_id	FK, Not Null	int	Subject code copied here to speed filtering and roll-ups.
department_id	FK, Not Null	int	Department owning the offering/faculty, copied for fast grouping.
faculty_user_id	FK, Not Null	int	The faculty member being evaluated.
form_template_id	FK, Not Null	int	The versioned evaluation form used for this period (weights/structure).
n_submissions	Not Null, Default 0	int	Count of completed evaluations included in this calculation.
n_valid_numeric	Not Null, Default 0	int	Number of usable Likert answers actually used (blanks/invalids excluded).
n_missing_numeric	Not Null, Default 0	int	Expected numeric answers minus valid ones; highlights data gaps.
quant_score_raw	Not Null	decimal(6,4)	The overall quantitative score before normalization (criterion means-of-medians combined by the form's weights).
mu_quant	Not Null	decimal(6,4)	The department average used to compare this result (baseline μ).
sigma_quant	Not Null	decimal(6,4)	How spread out the department's scores are (baseline σ). If 0, z=0.

cohort_type	Not Null, Default 'dept'	enum('dept','subject','global')	Identifies the comparison group for μ/σ. Policy: stored as 'dept'.
cohort_key_id	Nullable	int	The ID of that comparison group (e.g., department_id when 'dept').
cohort_n	Not Null, Default 0	int	How many results were used to compute μ/σ; helps interpret stability.
z_quant	Not Null	decimal(6,4)	Normalized quantitative score: how far above/below the department average this result is.
z_qual	Not Null	decimal(6,4)	The qualitative z-score for the same row (joined from SENTIMENT_AGGREGATES).
final_score_60_40	Not Null	decimal(6,4)	The combined result students and admins expect: 0.60×z_quant + 0.40×z_qual.
question_medians_json	Optional	JSON or TEXT	*(Optional)* Compact list of per-question medians for the breakdown page.
criterion_scores_json	Optional	JSON or TEXT	*(Optional)* Per-criterion “mean of medians” used in the weighted total.
criterion_weights_json	Optional	JSON or TEXT	*(Optional)* The criterion weights in effect (sum=100) for transparency.
calc_run_id	PK part, Not Null	int	The batch/run this row came from; new runs append new versions.

calc_version	Not Null	varchar(20)	Version tag of the calculation code/config used for this run.
source_snapshot_id	Not Null	int	Identifier for the raw data snapshot that was processed.
computed_at	Not Null	timestamp	When this record was computed (UTC).
is_final_snapshot	Not Null, Default false	boolean	Marks rows as frozen once the period closes; history won't be overwritten.

Table 35.1
SPECIAL TABLE CONSTRAINTS OF NUMERICAL_AGGREGATES TABLE

Constraint Name	**Type**	**Columns Involved**	**Description**
pk_num_aggs	PRIMARY KEY	(university_id, school_term_id, assessment_period, subject_offering_id, faculty_user_id, calc_run_id)	Ensures uniqueness per result and per calculation run.
chk_num_assessment_period	CHECK	assessment_period in ('midterms','finals')	Validates the period value.
chk_num_sigma_nonneg	CHECK	sigma_quant >= 0	Disallows negative standard deviation.

The tables above (refer to tables 35 and 35.1) hold the versioned quantitative results for a given teaching assignment and period, with enough context to support fair comparisons and reproducible analytics. Each row represents one calculation run for a class/section in a term (e.g., midterms or finals), including submission counts (valid/missing), raw indices, and standardized scores (z-scores) relative to a cohort baseline. Storing μ, σ, cohort identity/size, and calculation version makes the results self-describing, so scores can be explained and reproduced even if source data or algorithms change. Optional embedded breakdowns (per-question medians, criterion scores/weights) allow transparent drill-downs without reprocessing. A paired constraints object enforces correctness and comparability: a composite primary key prevents duplicate runs, assessment-period checks validate timing, and non-negative variance guards eliminate impossible dispersions. Together, this design supports dashboards (rankings, thresholds, eligibility checks), policy reporting (snapshots), and longitudinal studies while preserving auditability.
Table 36

DATA DICTIONARY OF SENTIMENT_AGGREGATES TABLE

Field Name	Constraints	Data Type	Description
university_id	FK, Not Null	int	University that owns the record.
school_term_id	FK, Not Null	int	The school year + semester this result belongs to.
assessment_period	Not Null	enum('midterms','finals')	Which portion of the term the result covers.
subject_offering_id	FK, Not Null	int	The exact class/section evaluated (the teaching assignment).
subject_id	FK, Not Null	int	Subject code copied here to speed filtering and roll-ups.
department_id	FK, Not Null	int	Department owning the offering/faculty, copied for fast grouping.
faculty_user_id	FK, Not Null	int	The faculty member being evaluated.
n_comments	Not Null, Default 0	int	Count of non-empty open-ended comments received.
n_valid_comments	Not Null, Default 0	int	Comments successfully analyzed by the NLP models (clean and supported).

avg_prob_positive	Not Null	decimal(6,5)	Average model probability for positive sentiment across valid comments.
avg_prob_neutral	Not Null	decimal(6,5)	Average model probability for neutral sentiment.
avg_prob_negative	Not Null	decimal(6,5)	Average model probability for negative sentiment.
prevailing_label	Not Null	enum('positive','neutral','negative')	The sentiment that dominates after averaging (what most comments lean to).
qual_score_raw	Not Null	decimal(6,4)	A single, human-friendly qualitative index (e.g., positive − negative, range ~[−1,1]) that summarizes the mood.
mu_qual	Not Null	decimal(6,4)	The department average of that qualitative index (baseline μ).
sigma_qual	Not Null	decimal(6,4)	How spread out the department's qualitative scores are (baseline σ). If 0, z=0.
cohort_type	Not Null, Default 'dept'	enum('dept','subject','global')	Identifies the comparison group for μ/σ. Policy: stored as 'dept'.

cohort_key_id	Nullable	int	The ID of that comparison group (e.g., department_id when 'dept').
cohort_n	Not Null, Default 0	int	How many results were used to compute μ/σ; helps interpret stability.
z_qual	Not Null	decimal(6,4)	Normalized qualitative score: how far above/below the department average this result is.
keywords_topn_json	Optional	JSON or TEXT	*(Optional)* Top key phrases with relevance, used for word clouds and "what changed" insights.
model_version	Not Null	varchar(40)	Version tags of the NLP components (e.g., XLM-R sentiment, KeyBERT).
calc_run_id	PK part, Not Null	int	The batch/run this row came from; new runs append new versions.
calc_version	Not Null	varchar(20)	Version tag of the calculation code/config used for this run.
source_snapshot_id	Not Null	int	Identifier for the raw data snapshot that was processed.

computed_at	Not Null	timestamp	When this record was computed (UTC).
is_final_snapshot	Not Null, Default false	boolean	Marks rows as frozen once the period closes; history won't be overwritten.

Table 36.1
SPECIAL TABLE CONSTRAINTS OF SENTIMENT_AGGREGATES TABLE

Constraint Name	**Type**	**Columns Involved**	**Description**
pk_sent_aggs	PRIMARY KEY	(university_id, school_term_id, assessment_period, subject_offering_id, faculty_user_id, calc_run_id)	Ensures uniqueness per result and per calculation run.
chk_sent_assessment_period	CHECK	assessment_period in ('midterms','finals')	Validates the period value.
chk_sent_prob_simplex	CHECK	(avg_prob_positive + avg_prob_neutral + avg_prob_negative) between 0.99 and 1.01	Sanity check that averaged probabilities sum to ≈1.
chk_sent_sigma_nonneg	CHECK	sigma_qual >= 0	Disallows negative standard deviation.

The tables above (refer to tables 36 and 36.1) captures the qualitative side of evaluations by summarizing open-ended comments into interpretable metrics and standardized comparisons. For each class/section and period, the table records volumes (comments received/valid), average sentiment probabilities, a human-friendly qualitative index, and the corresponding z-score against a cohort baseline—making it possible to track "mood" alongside numeric ratings. Versioning (model and calc) plus source snapshot IDs ensure results are reproducible and attributable even as NLP components improve over time. Optional top-keywords payloads power word clouds and "what changed" insights without rerunning heavy extraction. The paired constraints enforce statistical sanity (probabilities roughly summing to 1, non-negative dispersion), valid periods, and a composite key that distinguishes calculation runs. This structure

enables side-by-side fusion with numerical aggregates (e.g., 60/40 blends) while keeping the qualitative pipeline independently explainable and auditable.

Table 37

DATA DICTIONARY OF AI_SUGGESTIONS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
suggestion_id	PK	int	Unique ID.
university_id	FK	int	Tenant university.
evaluator_role	Not Null	enum (‘student’, ‘department_head’)	This field is to specify or tailor suggestions to the feedback source (e.g., whether coming from the students’ evaluation or department heads’ evaluations).
user_id	FK	int	The user receiving the suggestion/s (either a faculty or department head).
subject_id	FK	int	The subject that is related to a particular AI-generated suggestion (nullable for general).
school_term_id	FK	int	Academic term.
assessment_period	Not Null	Enum ('midterm,' 'finals')	Assessment period.

prompt_text	Not Null	text	Stores the full, structured prompt that was sent to the Gemini API to generate suggestion.
improvement_area	Not Null	varchar(100)	Area targeted for improvement (e.g., "Teaching Methods").
suggestion_text	Not Null	text	Text of AI-generated suggestion.
created_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Created timestamp.
updated_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Last update timestamp.

This table (Table 37) stores the textual AI-generated suggestions created by the Flan-T5 model for faculty improvement. Suggestions are tagged by improvement area and linked to faculty, academic term, and assessment period, enabling users to view actionable feedback in context. Recording confidence and accuracy metrics supports transparency and informed decision-making by faculty and administrators. This helps the system provide targeted, data-driven coaching recommendations.

Table 38

DATA DICTIONARY OF FLAGGED_EVALS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
flagged_evaluation_id	PK	int	Unique flag record ID.
submission_id	FK	int	Evaluation submission flagged.
university_id	FK	int	Tenant university.
flagged_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	When flagged.

flagged_by_user_id	FK	int	The user who raised the flagged status in which it is null if it is system-generated.
status	Not Null	enum('pending,' 'reviewed,' 'resolved,' 'ignored')	Flag review status.
reviewed_by_user_id	FK	int	Admin/user who reviewed the flag.
reviewed_at	Null	timestamp	When reviewed.
resolution_notes	Null	text	Notes on resolution.
resubmission_deadline	Null	timestamp	Set by the system when an admin requests a resubmission, marking the exact time the grace period ends.
created_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Creation timestamp.
updated_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Last update timestamp.

Table 39
DATA DICTIONARY OF FLAGGED_REASONS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
flagged_reason_id	PK	int	Unique ID.
flagged_evaluation_id	FK	int	Parent flagged evaluation.

reason_code	Not Null	varchar(100)	Code or short description (e.g., 'rating_sentiment_mismatch').
reason_description	Not Null	text	Detailed explanation.
created_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	When reason was recorded.
updated_at	Not Null, Default: CURRENT_TIMESTAMP	timestamp	Last update timestamp.

These tables (refer to Table 38 and Table 39) form a focused review ledger for potentially problematic evaluation submissions: the former captures the case—who/when it was flagged (including system flags), its lifecycle status, reviewer/timestamps, and any resolution note—anchored to the underlying submission and tenant for triage, SLA aging, and accountable outcomes; the latter stores append-only reasons as discrete entries with stable codes and human explanations so multiple signals can justify one case and pattern analytics stay clean. This separation lets the UI surface "what needs attention" at the case level while preserving drill-down into specific triggers, and it supports policy auto-flags, defensible decisions, and post-mortems without entangling evolving heuristics with the case lifecycle.

Table 40
DATA DICTIONARY OF NOTIFICATIONS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
notification_id	PK	int	Unique ID.
user_id	FK	int	Parent flagged evaluation.
notification_type	Not Null	varchar(100)	Code or short description (e.g., 'rating_sentiment_mismatch').
content	Not Null	text	Detailed explanation.
created_at	Not Null, Default: Current Timestamp	timestamp	When the notification was generated.

read_at	Null	timestamp	When the user marked the notification as read

This table (refer to table 40 above) delivers system-generated alerts to end users—typically reviewers, faculty, or admins—when notable events occur (e.g., anomalies, policy triggers, or workflow changes). Each notification carries a concise type code and a human-readable explanation, giving recipients both a machine handle for filtering and sufficient context to act. Read tracking supports UX (badges, inbox count) and metrics (time-to-read/acknowledge); creation timestamps enable chronological timelines and SLA analysis for critical alerts. In practice, this table powers in-app inboxes, bell icons, and email/Push fan-out, and serves as an auditable ledger of what the system surfaced and when—useful for support investigations and user accountability.

Table 41

DATA DICTIONARY OF BACKGROUND_JOBS TABLE

Field Name	**Constraints**	**Data Type**	**Description**
job_id	PK	int	Unique identifier for the asynchronous job record
job_uuid	Not Null; Unique	varchar(36)	The UUID assigned by the RQ worker for direct queue management.
university_id	FK; Not Null	int	The university tenant this job belongs to.
initiated_by_user_id	FK; Not Null	int	The user who started the job.
job_type	Not Null	enum('academic_structure_import', 'user_enrollment_import', 'evaluation_import', 'period_cancellation', 'report_generation')	The type of the asynchronous job (e.g., an academic structure import job, report generation, etc.).
status	Not Null; Default: queued	enum('queued', 'processing', 'completed', 'failed', 'cancelled')	The current lifecycle status of the job.
source_filename	Null	varchar(255)	The original name of the uploaded file for import jobs.

submitted_at	Not Null; Default: Current Timestamp	timestamp	Timestamp when the job was enqueued.
started_at	Null	timestamp	Timestamp when a worker began processing the job.
completed_at	Null	timestamp	Timestamp when the job finished successfully or failed.
error_details_path	Null	varchar(255)	Path to a detailed error report file for jobs with a ‘failed’ status.
result_file_path	Null	varchar(255)	Path to the output file for completed report generation jobs.

Finally, the table above (refer to table 41) serves as the system’s audit-ready ledger of all asynchronous tasks executed per university tenant—spanning bulk imports (academic structure, user enrollments, evaluations), period cancellations, and report generation. Each record ties the job to its tenant and initiator, classifies the work via a controlled `job_type`, and tracks lifecycle transitions (`queued → processing → completed/failed/cancelled`) with submitted, start, and completion timestamps to reconstruct exact execution history and compute durations. The table also stores the queue-facing `job_uuid` for correlation with Redis/RQ operations (e.g., monitoring, cancellation) and, when applicable, links to artifacts such as the original source file, generated outputs, and detailed error reports for post-mortem review. Together, these attributes enable operational dashboards, user-visible progress states, SLA analytics (success rates, throughput), and compliance-grade traceability across tenants without reprocessing raw logs.