**Database Driven Application - Study Abroad Database**

**1.- Database Topic proposal**

**Study Abroad Database**

This database will manage a study abroad program for a university, encompassing the process of applying to different available programs and obtaining information related to them.

**Components**

* Students
* Study Abroad Programs
* Program Sessions
* Applications
* User roles

**Non-trivial modeling aspects**

Three-way relationship: Student – Application – Session

**Features**

* User Access Control
* Program Browse
* Application Management
* Record Keeping
* Reporting

**2.- Specifications**

**Overview**

The Study Abroad Database is a system designed to manage the study abroad process at HWS, from student applications and program administration to post-program reports and summary statistics.

**User Groups**

Students, Study Abroad Office Staff, Program Administrators

**Data requirements**

Every user in this database will have a role allowing them access to different functionalities and information within the application (this user can be either a student, study abroad office staff, or a program administrator). Each user will have an ID and basic authentication credentials (username and password). Students have a unique ID, and personal information will be stored for them such as first and last name, email, phone number, and academic records such as completed credits, cumulative GPA, declared major, minor, relevant language proficiency, and academic standing. Programs will have an ID, specific name, type, destination, partner institution, program requirements (such as minimum GPA and/or language, and if so, which values for them), cost, and duration. A program that is of type faculty-led will have a faculty member associated with it. A program name together with a specific partner institution uniquely identifies a program. Sessions will have an ID, they will be linked to the program they belong to, and will have a start date, end date, and a capacity. A program ID together with a start date uniquely identifies a session. Finally, applications are uniquely identifiable by the student that submitted, and the session associated with the submission, and each will have a date and a status (submitted, under review, approved, waitlisted, rejected, withdrawn, deferred, accepted/declined by the student, in progress, completed, canceled). Applications also contain a student statement.

**Functional requirements**

**All users**

* Log in, log out of the system
* View and update profile/account information

**Students**

* Browse available study-abroad programs and view their details and requirements
* Submit study abroad applications
* Check application status

**Study Abroad Office Staff**

* Review and process student applications
* Generate reports on application and enrollment trends
* Manage own user accounts or students’

**Program Administrators**

* Manage program details and requirements
* Generate reports for specific programs

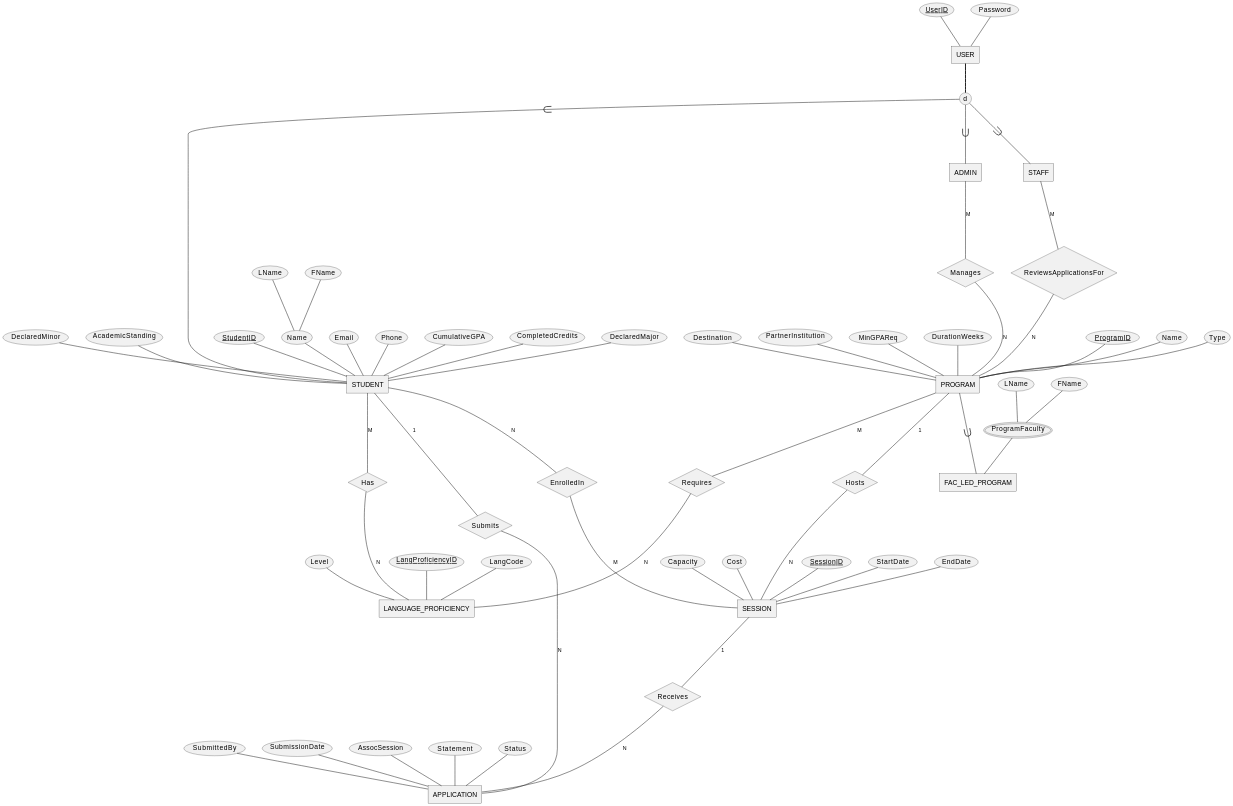
**Essential functionality**

User authentication and authorization, basic application submission and management, program listing and detail viewing, application status management, program eligibility checking (GPA, language requirements).

**Important functionality**

Program search and filtering, application management, analytics and reporting tools, application status review, user (student and non-student) profile management.

**3.- Conceptual Model**

****

**Policy constraints**

* Students can only apply to programs for which they meet the minimum GPA and language requirements.
* Application submission deadlines are based on program or session start dates.
* Enrollment in a session cannot exceed its capacity
* Staff members can only review applications for programs they are authorized to review
* Students cannot submit multiple applications for the same program in the same session
* Students can only enroll if their application has been approved/accepted

**Core constraints**

* Session start date must be before the end date
* Program GPA requirement must be between 0.0 and 4.0
* Session capacity and current enrollment must be non-negative integers
* User role must be one of student, staff, or admin

**Design analysis**

* I modeled sessions to be separate entities rather than being part of the program. This allows programs to have multiple sessions with different dates and capacities, providing flexibility.
* For student academic information, I included it directly in the student entity. This is because we will consider a single current value (at the time of the application) for each one.
* Enrollment tracking, as a relationship, allows tracking of which students are enrolled in which programs separate from the application process. This would help to manage current enrollments and historical data.

**4.- Database Design**

**Functional dependencies & candidate keys**

Based on the data requirements and the ER model, several functional dependencies decisions are already captured through stating ID attributes for entities. Here are other functional dependencies that are not captured by the ER diagram for entities which, although identifiable through combinations of attributes, have a given ID associated with them, and others that do not.

**PROGRAM**

{name, partnerInstitution} 🡪 {programId, type, destination, minGPAReq, durationWeeks}

{programId} 🡪 {name, type, destination, partnerInstitution, minGPAReq, cost, durationWeeks}

Candidate keys for PROGRAM:

* {name, partnerInstitution}
* {programId}

**SESSION**

{programId, startDate} 🡪 {sessionId, endDate, capacity}

{sessionId} 🡪 {programId, startDate, endDate, capacity}

Candidate keys for SESSION:

* {programId, startDate}
* {sessionId}

**APPLICATION**

{submittedBy, assocSession} 🡪 {submissionDate, status, statement}

Candidate keys for APPLICATION:

* {submittedBy, assocSession}
* {submittedBy, assocSession, submissionDate, status, statement}

**ER to relational schema**

**USER(userId, password)**

userId – VARCHAR(30), [PK]

password – CHAR(40), NOT NULL

**STUDENT(studentId, userId, fName, lName, email, phone, cumulativeGPA, completedCredits, declaredMajor, declaredMinor, academicStanding)**

studentId – INT, [PK]

userId – VARCHAR(30) [FK] (STUDENT.userId 🡪 USER.usedId)

fName – VARCHAR(50), NOT NULL

lName – VARCHAR(50), NOT NULL

email – VARCHAR(100), NOT NULL, UNIQUE

phone – VARCHAR(15), NOT NULL, UNIQUE (Flexibility for international numbers)

cumulativeGPA – DECIMAL(3,2), CHECK (cumulativeGPA >= 0.0 AND cumulativeGPA <= 5.0), NOT NULL,

completedCredits – DECIMAL(3,1), CHECK (completedCredits >= 0), NOT NULL

declaredMajor – VARCHAR(50)

declaredMinor – VARCHAR(50)

academicStanding – ENUM(‘Good Standing’, ‘Academic Warning’, ‘Academic Probation’, ‘Suspended’), NOT NULL

UNIQUE (userId)

**MANAGES(adminId, programId)**

PRIMARY KEY (adminId, programId)

adminId – VARCHAR(30) [FK] (MANAGES.adminId 🡪 USER.userId)

programId – INT [FK] (MANAGES.programId 🡪 PROGRAM.programId)

TRIGGER 🡪 Prevent non-admins from managing programs, checks NOT EXISTS in STUDENT and REVIEWS\_APPLICATIONS\_FOR

**REVIEWS\_APPLICATIONS\_FOR(staffId, programId)**

PRIMARY KEY (staffId, programId)

staffId – VARCHAR(30) [FK] (REVIEWS\_APPLICATIONS\_FOR.staffId 🡪 USER.userId)

programId – INT [FK] (REVIEWS\_APPLICATIONS\_FOR.programId 🡪 PROGRAM.programId)

TRIGGER 🡪 Prevent non-staff from reviewing applications, checks NOT EXISTS in STUDENT and MANAGES

**PROGRAM(programId, name, type, destination, partnerInstitution, minGPAReq, durationWeeks)**

programId – INT [PK]

name – VARCHAR(100), NOT NULL

type – ENUM(‘Exchange’, ‘Faculty-Led’, ‘Third-Party’), NOT NULL

destination – VARCHAR(100), NOT NULL

partnerInstitution – VARCHAR(100), NOT NULL

minGPAReq – DECIMAL(3,2), CHECK (minGPAReq >= 0.0 AND minGPAReq <= 5.0)

durationWeeks – INT, CHECK (durationWeeks > 0), NOT NULL

UNIQUE (name, partnerInstitution)

**PROGRAM\_FACULTY(programId, fName, lName)**

PRIMARY KEY (programId, fName, lName)

programId – INT [FK] (PROGRAM\_FACULTY.programId 🡪 PROGRAM.programId)

fName – VARCHAR(50), NOT NULL

lName – VARCHAR(50), NOT NULL

**SESSION(sessionId, programId, startDate, endDate, capacity, cost)**

sessionId – INT [PK]

programId – INT [FK] (SESSION.programId 🡪 PROGRAM.programId)

startDate – DATE, NOT NULL

endDate – DATE, NOT NULL

capacity – INT, CHECK (capacity > 0), NOT NULL

cost – DECIMAL(10,2) CHECK (cost >= 0), NOT NULL

UNIQUE (programId, startDate)

CHECK (endDate > startDate)

CHECK (DATEDIFF(‘week’, startDate, endDate) = (SELECT durationWeeks FROM PROGRAM WHERE PROGRAM.programId = SESSION.programId)))

**LANGUAGE\_PROFICIENCY(langProficiencyId, langCode, level)**

langProficiencyId – INT [PK]

langCode – VARCHAR(10), NOT NULL

level – ENUM(‘Beginner’, ‘Intermediate’, ‘Advanced’), NOT NULL

**STUDENT\_LANG\_PROF(studentId, langProficiencyId)**

PRIMARY KEY (studentId, langProficiencyId)

studentId – INT [FK] STUDENT\_LANG\_PROF.studentId 🡪 STUDENT.studentId)

langProficiencyId – INT [FK] STUDENT\_LANG\_PROF.langProficiencyId 🡪 LANGUAGE\_PROFICIENCY.langProficiencyId)

**PROGRAM\_LANG\_REQ(programId, langProficiencyId)**

PRIMARY KEY (programId, langProficiencyId)

programId – INT [FK] PROGRAM\_LANG\_REQ.programId 🡪 PROGRAM.programId)

langProficiencyId – INT [FK] PROGRAM\_LANG\_REQ.langProficiencyId 🡪 LANGUAGE\_PROFICIENCY.langProficiencyId)

**ENROLLED\_IN(studentId, sessionId)**

PRIMARY KEY (studentId, sessionId)

studentId – INT [FK] (STUDENT.studentId 🡪 ENROLLED\_IN.studentId)

sessionId – INT [FK] (SESSION.sessionId 🡪 ENROLLED\_IN.sessionId)

**APPLICATION(submittedBy, assocSession, submissionDate, statement, status)**

PRIMARY KEY (submittedBy, assocSession)

submittedBy – INT [FK] (APPLICATION.submittedBy 🡪 STUDENT.studentId), NOT NULL

assocSession – INT [FK] (APPLICATION.assocSession 🡪 SESSION.sessionId), NOT NULL

submissionDate – TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

statement – TEXT(5000), NOT NULL

status – ENUM(‘submitted’, ‘under\_review’, ‘approved’, ‘waitlisted’, ‘rejected’, ‘withdrawn’, ‘deferred’, ‘accepted\_by\_student’, ‘declined\_by\_student’, ‘in\_progress’, ‘completed’, ‘canceled’), NOT NULL

TRIGGER 🡪 Ensures only students can submit applications, checks EXISTS in STUDENT

**Foreign Key Dependencies**

* STUDENT.userId 🡪 USER.userId
* MANAGES.adminId 🡪 USER.userId
* MANAGES.programId 🡪 PROGRAM.programId
* REVIEWS\_APPLICATIONS\_FOR.staffId 🡪 USER.userId
* REVIEWS\_APPLICATIONS\_FOR.programId 🡪 PROGRAM.programId
* PROGRAM\_FACULTY.programId 🡪 PROGRAM.programId
* SESSION.programId 🡪 PROGRAM.programId
* STUDENT\_LANG\_PROF.studentId 🡪 STUDENT.studentId
* STUDENT\_LANG\_PROF.langProficiencyId 🡪 LANGUAGE\_PROFICIENCY.langProficiencyId
* PROGRAM\_LANG\_REQ.programId 🡪 PROGRAM.programId
* PROGRAM\_LANG\_REQ.langProficiencyId 🡪 LANGUAGE\_PROFICIENCY.langProficiencyId
* ENROLLED\_IN.studentId 🡪 STUDENT.studentId
* ENROLLED\_IN.sessionId 🡪 SESSION.sessionId
* APPLICATION.submittedBy 🡪 STUDENT.studentId
* APPLICATION.assocSession 🡪 SESSION.sessionId

**Normalization**

**USER(userId, password)**

**STUDENT(studentId, userId, fName, lName, email, phone, cumulativeGPA, completedCredits, declaredMajor, declaredMinor, academicStanding)**

**MANAGES(adminId, programId)**

**REVIEWS\_APPLICATIONS\_FOR(staffId, programId)**

**PROGRAM(programId, name, type, destination, partnerInstitution, minGPAReq, durationWeeks)**

**PROGRAM\_FACULTY(programId, fName, lName)**

**SESSION(sessionId, programId, startDate, endDate, capacity, cost)**

**LANGUAGE\_PROFICIENCY(langProficiencyId, langCode, level)**

**STUDENT\_LANG\_PROF(studentId, langProficiencyId)**

**PROGRAM\_LANG\_REQ(programId, langProficiencyId)**

**ENROLLED\_IN(studentId, sessionId)**

**APPLICATION(submittedBy, assocSession, submissionDate, statement, status)**

The schema is in BCNF because it satisfies all preceding normal forms, and every non-trivial functional dependency is determined by a superkey. It is in 1NF as all attributes contain atomic values and there are no repeating groups. It is in 2NF because it is in 1NF and all non-key attributes are fully functionally dependent on their primary keys (e.g. in STUDENT, attributes like fName, email depend on the entire studentId). It is in 3NF because it is in 2NF and there are no transitive dependencies (no non-key attributes depend on other non-key attributes). Finally, it is in BCNF because all determinants are candidate keys (e.g. in STUDENT, besides studentId determining all attributes, the UNIQUE constraints on userId makes it a candidate key). All relations follow this pattern, therefore ensuring that BCNF was reached throughout the schema.

**Design Discussion**

First, I included every entity represented on the ER model and preserved their attributes. To handle STAFF and ADMIN entities, instead of using a type attribute in USER table, I implemented database triggers in MANAGES and REVIEWS\_APPLICATIONS\_FOR tables to ensure not only that students cannot participate in these relationships, but also that admins and staff roles remain mutually exclusive. These triggers actively prevent role conflicts by checking existing records before inserting, maintaining clear role separation while simplifying the schema. The USER table serves as a base authentication table, with STUDENT having a one-to-one relationship through a UNIQUE userId constraint. VARCHAR type was used for storing strings, with lengths determined by considering common usage while allowing flexibility. For language proficiency possession for students and requirements for programs, both were represented as tables that relate to the LANGUAGE\_PROFICIENCY entity. I decided to create this entity to facilitate language level-code matching and allowing for further languages to be added instead of an ENUM approach. Status in application was addressed using ENUM, allowing for the database requirements to be met. To represent faculty-led programs, the implementation of PROGRAM\_FACULTY relation allows identification of those programs through programId as part of its primary key, along with the faculty member details (fName, lName), capturing multiple faculty members per program. Finally, functional dependencies are enforced by UNIQUE constraints and proper foreign key relationships, ensuring data integrity throughout the schema.

**Application Design – Front End**

Depending on the user’s role (determined by their presence in STUDENT, MANAGES, or REVIEWS\_APPLICATIONS\_FOR tables), different scenes and functionalities will be available. A main stage will display a simple login screen with username and password fields. Upon successful login, the application determines the user’s role through database queries and displays the appropriate dashboard. The student dashboard contains four main buttons: “View Programs”, “My Applications”, “View Profile and “Logout”. The “view programs” scene displays a simple table view of available programs. When trying to apply to programs, students can only successfully send their application to those where they meet the requirements (GPA and academic standing, and that have enough capacity). The application process involves a simple form with a text area for their statement. Students can view their submitted applications’ status through the “my applications” section, displayed in a table format. For admin users, their dashboard presents them with two buttons: “Manage Programs” and “Logout”. In the “manage programs” scene, they can visualize available programs, but they can only successfully apply edits to the ones they are allowed to manage (enforced by the database). Staff members have access to two buttons in their dashboard: “Review Applications”, and “Logout”. In the “review applications” interface, they can only see applications from students which have applied for programs where they can review for. All interfaces use simple tables and forms instead of complex grid layouts or images, focusing on core functionality and data management.

**Application Design – Back End**

**Database Connections**

As specified in the project instructions for this specific section, username\_user account will be used for view-based queries (SELECT only), whereas username\_admin will be used for stored procedure execution and data modifications. Access control is managed through view prefixes for regular user access, stored procedures for all data modifications, role verification within stored procedures, and separate connection handling for user/admin operations.

**Essential Views (all prefixed with view\_ for \_user account access):**

* User role (returns userId, role): Determines user’s role (student/admin/staff) based on table relationships
* Available programs (returns program details, spots available): Shows active programs with open sessions and capacity
* Student applications (returns application and program details): Displays student’s applications status and information
* Pending reviews (returns application and student details): Lists applications needing staff review
* Student profile (returns student academic information): Presents student personal and academic details
* Program requirements (returns program and language requirements): List program prerequisites

**Stored Procedures (requiring \_admin account access):**

* User login (parameters: userId, password, returns role): Authenticates users and determines their role by checking presence in user role view
* Submit application (parameters: studentId, sessionId, statement, returns boolean): Creates new application with eligibility checks and capacity verification, using REPEATABLE READ isolation to prevent conflicts
* Process application (parameters: applicationId, status, reviewerId, returns boolean): Updates application status after verifying reviewer authorization, using READ COMMITED isolation and row-level locking to prevent concurrent reviews
* Enroll student (parameters: studentId, sessionId, return boolean): Enrolls a student in the session they have accepted their offer for, changing the application status to ‘accepted\_by\_student’
* Update program details (parameters: programId, program details, returns boolean): Modifies program information

**Transactions and Concurrency Control:**

* Application submission uses REPEATABLE READ isolation with row-level locking on SESSION to prevent enrollment conflicts
* Application processing uses READ COMMITTED isolation with row-level locking on APPLICATION to prevent concurrent reviews
* Enroll student uses REPEATABLE READ isolation with row-level locking on SESSION to prevent concurrent modifications of the same session or application records
* All procedures include proper error handling and authorization verification