1) INTRODUCTION

1.1) Main Scenario:

An exchange student applies to a department of a university within the Erasmus+ programme. The purpose of this project is to provide course equivalency information by comparing the courses the student plans to take at the host university with the courses from the student's home university, in order to determine whether the courses are equivalent.

1.2) Rules:

1) There is an exchange student with a unique student no, major, address, phone, birth date and a name consisting of first, middle and last name. A student must have a first and last name.

2) An exchange student must belong to a department of a University in order to apply for Erasmus+ programme, this must be tracked by the database. A department can have many students.

3) An exchange student can apply to many departments and departments can have many applicants. But an applicant cannot apply to a school within the same country. The country information will be tracked within each department to ensure this constraint.

4) An erasmus application must store a unique application id, application date and a status. Status has four states: Pending, Approved, Rejected, Conditional.

5) A course can have multiple equivalent courses. Equivalency has multiple parameters: Difference of each course's ECTS credit, Language match check to see if language of the courses are the same. The values are calculated automatically. The equivalency status is decided by the host department

2) ANALYSIS OF DEPARTMENTS

2.1) Ege University – Computer Engineering:

Requirements:

1) The university has a unique id, name, phone, email and a location. A university is composed of multiple departments.

2) A department has a unique department id, name, location, multiple phone numbers and fax numbers and a level attribute indicating the main academic level associated with the department (e.g., BSc, MSc, PhD). A department can employ instructors and offer a curriculum.

3) An instructor has a unique id and email, name, phone and a rank indicating the current position of the instructor. An instructor can also have education information at the BSc, MSc, and PhD levels. An instructor can direct a curriculum. An instructor can teach multiple sections.

4) A curriculum has a unique id, a model of study indicating whether it is full-time or part-time, a release time, and a total ECTS value representing the sum of the courses contained in the curriculum. A curriculum is composed of multiple courses.

5) A course has a unique id, name, teaching language, objective, semester in which the course is offered, and instructional hours divided into theoretical, practical, and laboratory hours. A course also has an ECTS credit value and a delivery mode indicating whether it is face-to-face or online. A course can have multiple textbooks with name and edition information, learning outcomes, and multiple prerequisite courses.

6) A course can be of two types: mandatory or elective. Elective courses have a group number and can belong to one of several categories, including Technical, University, and Pedagogy.

7) A course can have multiple sections. A section has a unique id, section number, day on which it takes place, classroom number, and building number of the classroom.

8) A course has an assessment criteria. Assessment has a unique no, a term percentage and an end-term percentage, which together sum to 100. An assessment also has multiple components. The term and end-term percentages act as multipliers for the corresponding components.

9) An assessment has multiple components. Each component has a type (term or end-term), name, quantity, and weight. The total weight of all components of the same type must sum to 100.

10) A course has detailed content. Each content entry has a unique id, week number, weekly subject, learning method, practice method, and a preparatory explanation for each week.

2.2) Muğla Sıtkı Koçman University – Software Engineering

Requirements:

1) The university has a unique id, name, phone, email and a location. A university is composed of multiple departments.

2) A department has a unique department id, name, location, multiple phone numbers and fax numbers and a level attribute indicating the main academic level associated with the department (e.g., BSc, MSc, PhD). A department can employ instructors and offer a curriculum.

3) An instructor has a unique id and email, name, phone and a rank indicating the current position of the instructor. An instructor can also have education information at the BSc, MSc, and PhD levels. An instructor can direct a curriculum. An instructor can teach and assist multiple sections.

4) A curriculum has a unique id, a mode of study indicating whether it is full-time or part-time, a release time, and a total ECTS value representing the sum of the courses contained in the curriculum. A curriculum is composed of multiple courses.

5) A course has a unique id, name, teaching language, objective, semester in which the course is offered, and hours divided into theoretical, practical, and laboratory hours. A course also has an ECTS credit value and a delivery mode indicating whether it is face-to-face or online. A course can have multiple textbooks with name and edition information, as well as learning outcomes.

6) A course can be of two types: mandatory or elective.

7) A course can have multiple sections. A section has a unique id, section number, the day on which the section takes place, classroom number, and building number.

8) A course has an assessment criteria. Assessment has a unique no. An assessment also has multiple components.

9) Each component has a name, quantity, and weight. The sum of all component weights must be 100.

10) A course has detailed content. Each content entry has a unique id, week number, weekly subject, learning method of the week, and a preparatory explanation for each week.

2.3) Sakarya University – Information Systems Engineering

Requirements:

1) The university has a unique id, name, phone, email and a location. A university is composed of multiple departments.

2) A department has a unique department ID, name, location, multiple phone numbers, and a level attribute indicating the main academic level associated with the department (e.g., BSc, MSc, PhD). A department can employ instructors and offer a curriculum.

In addition, a department can assist a section with assistant instructors.

3) An instructor has a unique ID, email, name, phone number, and a rank that states the instructor's current position. An instructor can also have education information at the BSc, MSc, and PhD levels. An instructor can direct multiple courses and teach multiple sections.

4) A curriculum has a unique ID, a model of study that states whether the curriculum is full-time or part-time, the release time of the curriculum, and the total ECTS number of courses contained in the curriculum. A curriculum is composed of multiple courses.

5) A course has a unique ID, name, teaching language, objective, the semester in which the course takes place, ECTS credit, and hours divided into theoretical and practical hours. A course also has a delivery mode that is either face-to-face or online. A course can have multiple textbooks with a name and edition, as well as learning outcomes. Outcomes also have an assessment method and a learning method.

6) A course has two types: mandatory and elective. A mandatory course has three types: Faculty, YÖK, and Department. An elective course also has two types: Faculty and University.

7) Courses can have multiple sections. Sections have a unique ID, section number, the day on which the section takes place, the classroom number, and the building number of the classroom.

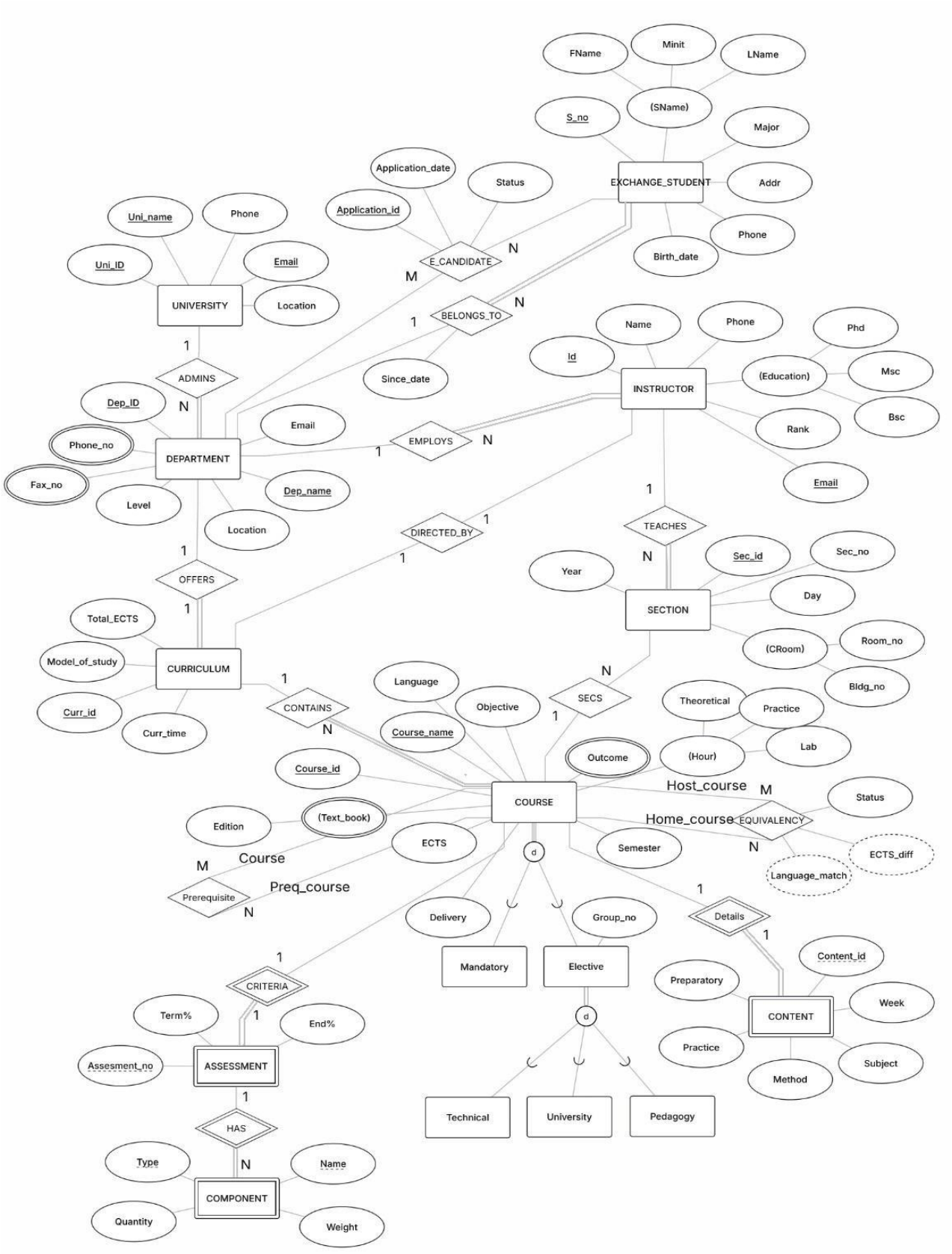
8) A course has an assessment criteria. Assessment has a unique no. An assessment also has multiple components.

9) Components have names, quantities, and weights. All weights must sum to 100.

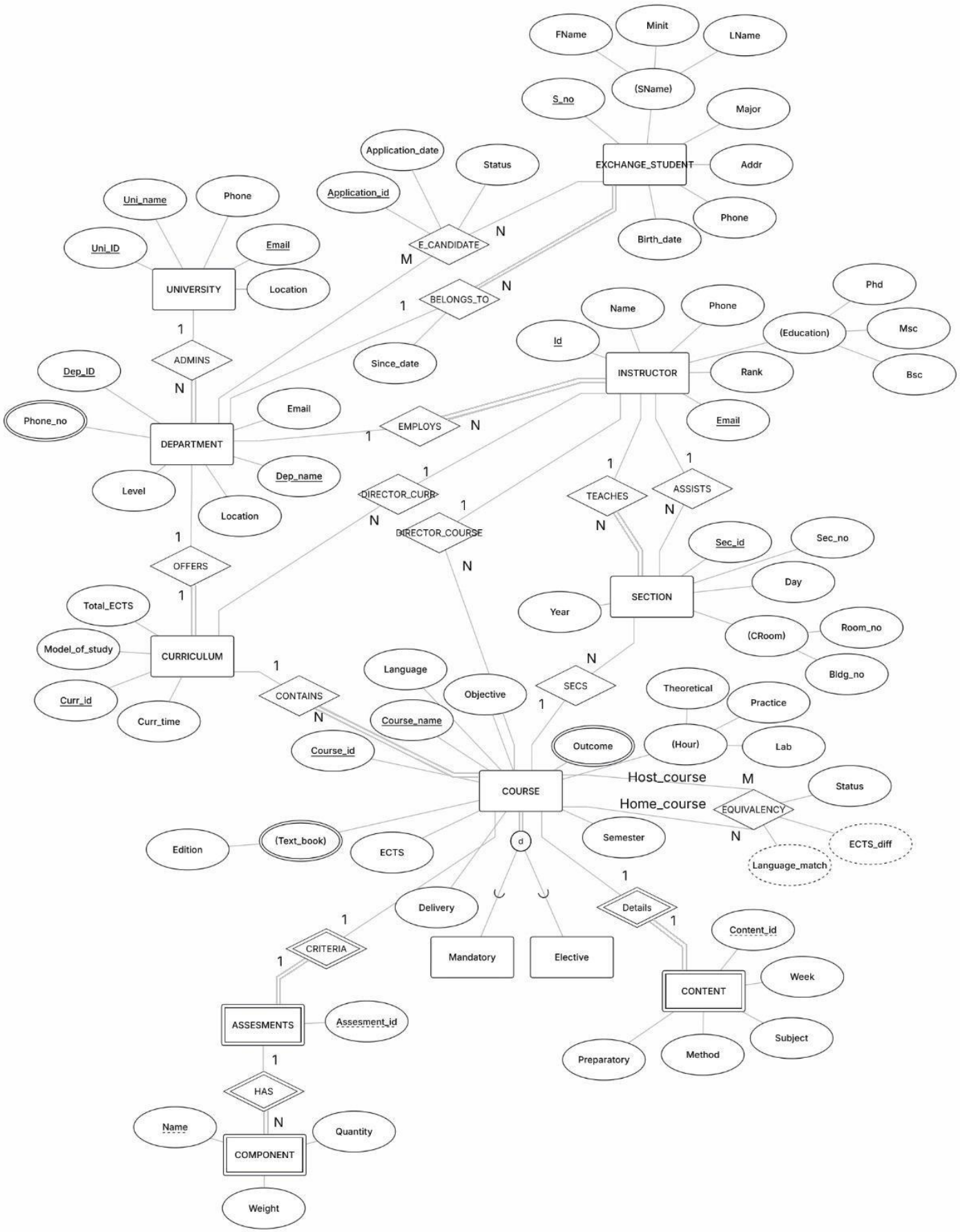
10) A course also has detailed content with a unique ID, week number, week subject, and a preparatory explanation for each week.

3) DESIGN - CONCEPTUAL MODELS

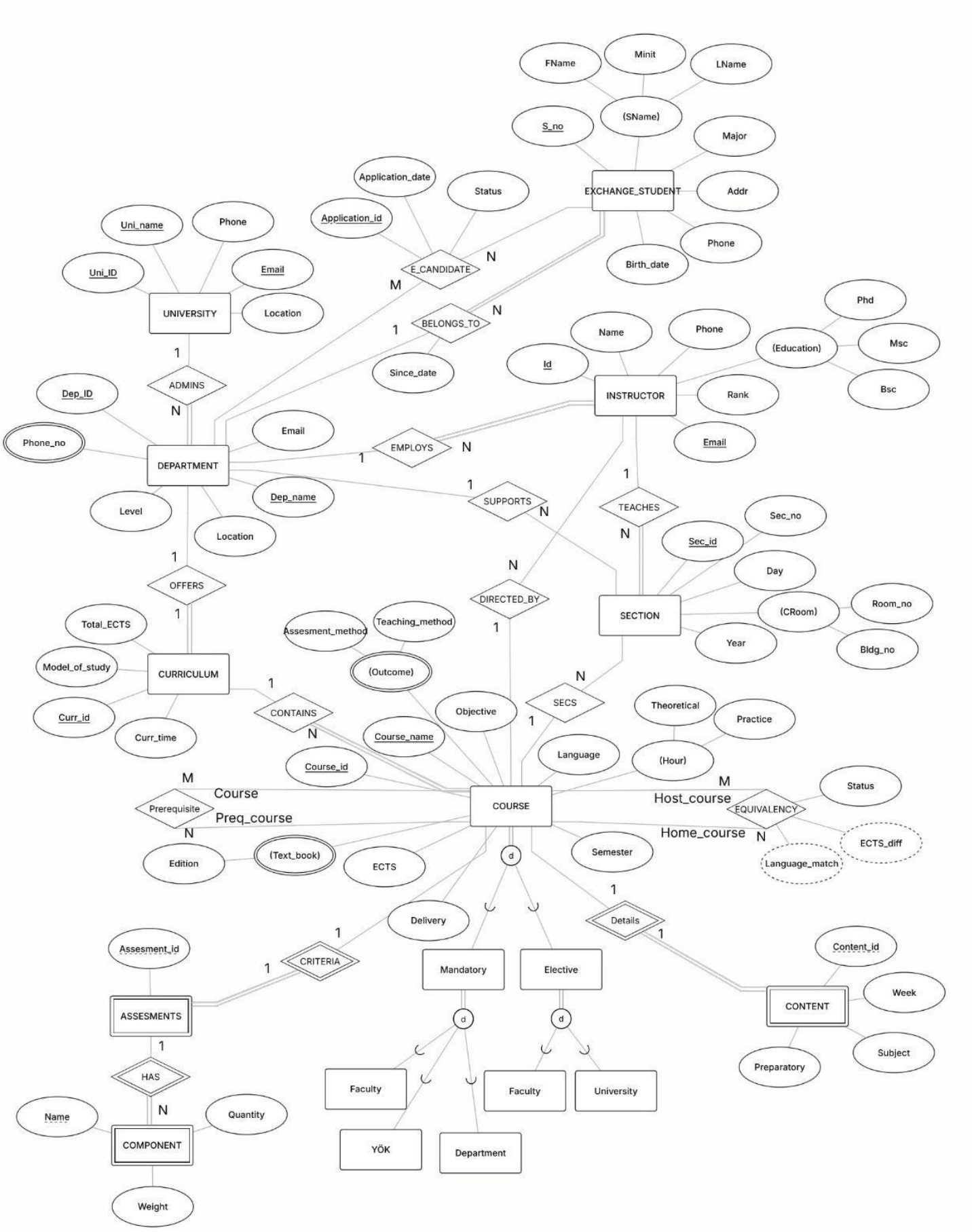
3.1) Ege University



3.2) Muğla Sıtkı Koçman University

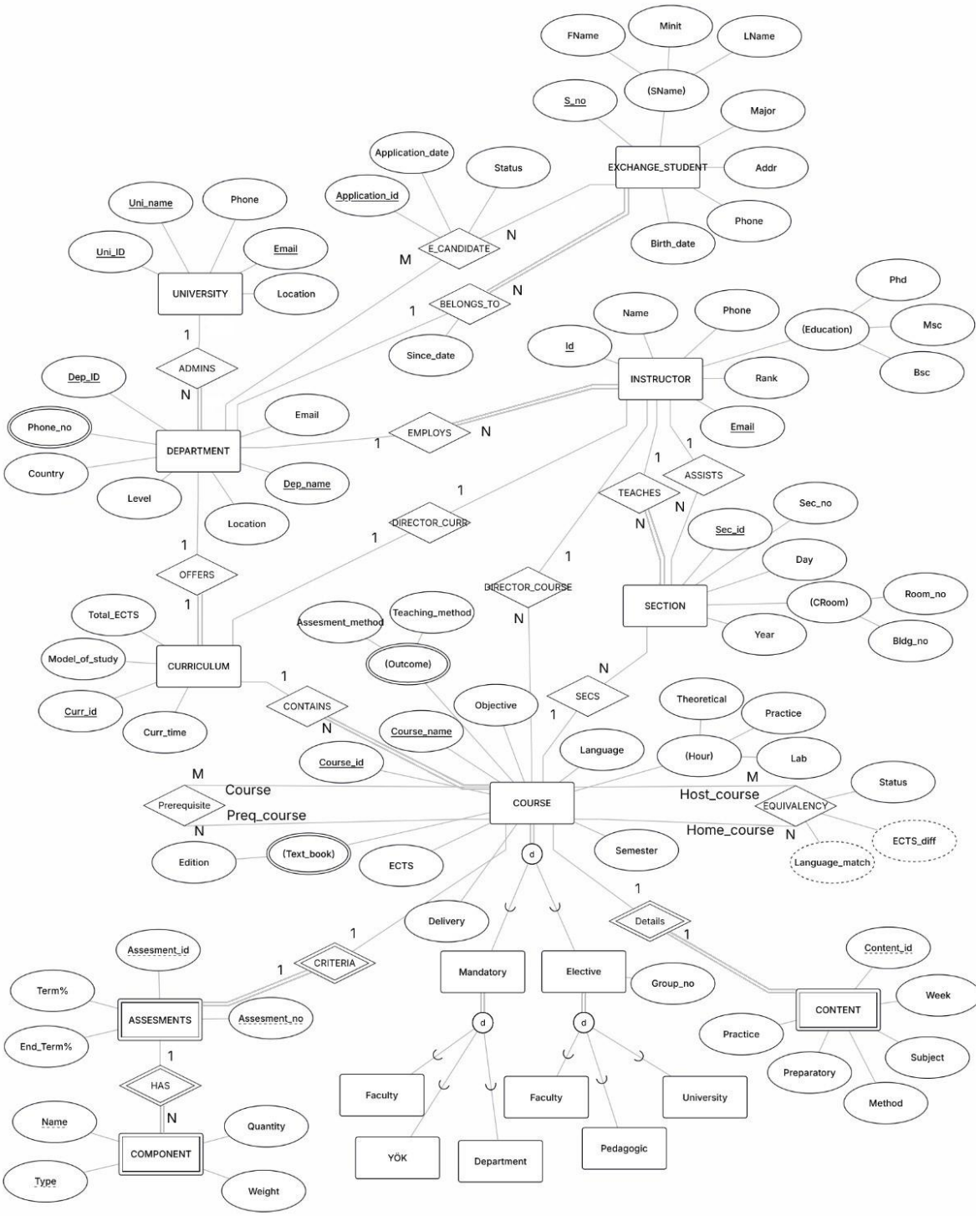


3.3) Sakarya University



3.4) Integrated EER Model

The three EER diagrams have been integrated into a single diagram by merging common entities. Attributes that were unique to each source diagram were added in order to preserve all original information. The ‘SUPPORTS’ relationship from the Sakarya University diagram was removed; instead, assistant information is now stored within the ‘INSTRUCTOR’ entity as an attribute representing instructors of a specific department. All other distinct relationships were retained to ensure complete data representation.



4) DESIGN - LOGICAL MODEL

1-)

- UNIVERSITY (Uni_id(PK), Uni_name, Phone, Email, Location)
- DEPARTMENT (Dep_id(PK), Location, Country, Dep_name, Email, Level)
- CURRICULUM (Curr_id(PK), Curr_time, Total_ects, Model-of-study)
- EXCHANGE-STUDENT (S_no(PK), Sname_Fname, Sname_Mname, Sname_Lname, Major, Addr, Phone, DOB)
- INSTRUCTOR (ID(PK), Name, Phone, Email, Rank, Education_Bsc, Education_Msc, Education_Phd)
- COURSE (Course_id(PK), Course_name, Semester, ECTS, Objective, Language, Delivers, Hour_Theo, Hour_Lab, Hour_Prac)
- SECTION (Sec_id(PK), Sec_no, Year, Day, CRoom_Bldg_no, CRoom_Room_no)

2-)

- ASSESSMENT (COURSE.Course_id(FK,PK), Assesment_id(PK), Term%, End-Term%) // via CRITERIA
- CONTENT (COURSE.Course_id(FK,PK), Content_id(PK), Week, Subject, Document, Method, Preparatory) // via DETAILS

3-)

- CURRICULUM (..., DEPARTMENT.Dep_id (FK)) // via OFFERS
- CURRICULUM (..., INSTRUCTOR.Dir_id(FK)) //via DIRECTOR_CURR

4-)

- DEPARTMENT (..., UNIVERSITY.Uni_id (FK)) // via ADMINS
- EXCHANGE_STUDENT (..., DEPARTMENT.Dep_id (FK), BELONGS_TO.Since_date) // via BELONGS_TO
- INSTRUCTOR (..., DEPARTMENT.Dep_id (FK)) // via EMPLOYS

- COURSE (..., INSTRUCTOR.Dir_id (FK)) // via DIRECTOR_COURSE
- COURSE (..., CURRICULUM.Curr_id (FK)) // via CONTAINS
- SECTION (..., COURSE.Course_id (FK)) // via SECS
- SECTION (..., INSTRUCTOR.Teacher_id (FK)) // via TEACHES
- SECTION (..., INSTRUCTOR.Assistant_id (FK)) // via ASSISTS

5-)

- E-CANDIDATE (DEPARTMENT.Dep_id(FK,PK), EXCHANGE_STUDENT.S_no(FK,PK), App_id(PK), App_date, Status)

6-)

- TEXTBOOK (COURSE.Course_id(FK,PK), Textbook, Edition) // via COURSE
- OUTCOME (COURSE.Course_id(FK,PK), Outcome, Assessment_Method, Learning_Method) // via COURSE
- PHONE_NO (DEPARTMENT.Dep_id(FK,PK), Phone-no) // via DEPARTMENT
- FAX_NO (DEPARTMENT.Dep_id(FK,PK), Fax-no) // via DEPARTMENT

7-)

- PREREQUISITE (COURSE.Course_id(FK,PK), COURSE.Prerequisite_id(FK,PK))
- EQUIVALENCY (COURSE.Host_Course_id(FK,PK), COURSE.Home_Course_id(FK,PK), Status)

8-) 8A:

- MANDATORY (COURSE.Course_id(FK,PK))
- ELECTIVE (COURSE.Course_id(FK,PK), Group-no)

9-) -

Second Iteration (Disjoint Chain / Weak entity Chain)

2:

- COMPONENT (ASSESMENT.Assesment_no(FK,PK), COURSE.Course_id(FK,PK), Name, Type, Weight, Quantity) //via HAS

8: - 8C:

- MANDATORY (..., MandatoryType)
- ELECTIVE (..., ElectiveType)

Resulting Relational Database Schema:

UNIVERSITY(Uni_id, Uni_name, Phone, Email, Location)

DEPARTMENT (Dep_id, Location, Country, Dep_name, Email, Level, Uni_id)

INSTRUCTOR(ID, Name, Phone, Email, Rank, Education_Bsc, Education_Msc, Education_Phd, Dep_id)

CURRICULUM (Curr_id, Curr_time, Total_ects, Model-of-study, Dep_id, Dir_id)

EXCHANGE-STUDENT(S_no, Sname_Fname, Sname_Mname, Sname_Lname, Major, Addr, Phone, DOB, Dep_id, Since_date)

COURSE(Course_id, Course_name, Semester, ECTS, Objective, Language, Delivers, Hour_Theo, Hour_Lab, Hour_Prac,Curr_id ,Dir_id,)

SECTION(Sec_id, Sec_no, Year, Day, CRoom_Bldg_no, CRoom_Room_no,Course_id, Teacher_id ,Assistant_id)

ASSESSMENT(Course_id, Assessment_id, Term%, End-Term%)

COMPONENT (Assesment_no, Course_id, Name, Type, Weight, Quantity)

CONTENT (Course_id, Content_id, Week, Subject, Document, Method, Preparatory)

E-CANDIDATE (Dep_id, S_no, App_id, App_date, Status)

PREREQUISITE (Course_id, Prerequisite_id)

EQUIVALENCY (Host_Course_id, Home_Course_id, Status)

MANDATORY (Course_id, MandatoryType)

ELECTIVE (Course_id, Group-no, ElectiveType)

TEXTBOOK (Course_id, Textbook, Edition)

OUTCOME (Course_id, Outcome, Assessment_Method, Learning_Method)

PHONE_NO (Dep_id, Phone_no)

FAX_NO (Dep_id, Fax_no)

5) SQL IMPLEMENTATION AND PHYSICAL DESIGN

5.1) DDL Script

This section presents the Data Definition Language (DDL) commands used to construct the database schema. The physical design translates the relational schema into actual SQL tables. To ensure data integrity and enforce business rules, various constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, and CHECK have been implemented.

Additionally, ON DELETE CASCADE actions have been defined for dependent entities (such as weak entities and multi-valued attributes) to prevent orphan records and maintain database consistency.

Below is the complete SQL script used to generate the database:

```
-- 1. UNIVERSITY & DEPARTMENT STRUCTURE

CREATE TABLE UNIVERSITY (
    Uni_id INT PRIMARY KEY,
    Uni_name VARCHAR(100) UNIQUE,
    Phone VARCHAR(20),
    Email VARCHAR(100) UNIQUE,
    Location VARCHAR(150)
```

```
);

CREATE TABLE DEPARTMENT (
    Dep_id INT PRIMARY KEY,
    Dep_name VARCHAR(100),
    Location VARCHAR(150),
    Country VARCHAR(20),
    Email VARCHAR(100),
    Level_Val VARCHAR(50),
    Uni_id INT NOT NULL,
    FOREIGN KEY (Uni_id) REFERENCES UNIVERSITY(Uni_id)
);

-- Multivalued Attribute: Department Phones
CREATE TABLE DEP_PHONE_NO (
    Dep_id INT,
    Phone_no VARCHAR(20),
    PRIMARY KEY (Dep_id, Phone_no),
    FOREIGN KEY (Dep_id) REFERENCES DEPARTMENT(Dep_id) ON DELETE CASCADE
);

-- Multivalued Attribute: Department Faxes
CREATE TABLE DEP_FAX_NO (
    Dep_id INT,
    Fax_no VARCHAR(20),
    PRIMARY KEY (Dep_id, Fax_no),
    FOREIGN KEY (Dep_id) REFERENCES DEPARTMENT(Dep_id) ON DELETE CASCADE
);

-- 2. ACADEMIC STAFF & CURRICULUM
CREATE TABLE INSTRUCTOR (
    Instructor_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Phone VARCHAR(20),
    Email VARCHAR(100),
    Rank_Val VARCHAR(50),
    Edu_Bsc VARCHAR(100),
    Edu_Msc VARCHAR(100),
    Edu_Phd VARCHAR(100),
    Dep_id INT NOT NULL,
    FOREIGN KEY (Dep_id) REFERENCES DEPARTMENT(Dep_id)
);

CREATE TABLE CURRICULUM (
    Curr_id INT PRIMARY KEY,
    Curr_time INT,
    Total_ects INT,
    Model_of_study VARCHAR(50),
    Dep_id INT NOT NULL,
    Director_id INT,
    FOREIGN KEY (Director_id) REFERENCES INSTRUCTOR(Instructor_ID),
    FOREIGN KEY (Dep_id) REFERENCES DEPARTMENT(Dep_id)
);

-- 3. COURSES & INHERITANCE
CREATE TABLE COURSE (
    Course_id VARCHAR(20) PRIMARY KEY,
    Course_name VARCHAR(100),
    Semester INT,
```

```
    ECTS INT,
    Objective TEXT,
    Language VARCHAR(50),
    Delivers TEXT,
    Hour_Theo INT,
    Hour_Lab INT,
    Hour_Prac INT,
    Curr_id INT NOT NULL,
    Director_id INT,
    FOREIGN KEY (Director_id) REFERENCES INSTRUCTOR(Instructor_ID),
    FOREIGN KEY (Curr_id) REFERENCES CURRICULUM(Curr_id),
    CONSTRAINT chk_ects_valid CHECK (ECTS > 0)
);

-- Subtype: Mandatory Course
CREATE TABLE MANDATORY (
    Course_id VARCHAR(20) PRIMARY KEY,
    MandatoryType VARCHAR(20),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id) ON DELETE CASCADE
);

-- Subtype: Elective Course
CREATE TABLE ELECTIVE (
    Course_id VARCHAR(20) PRIMARY KEY,
    Group_no INT,
    ElectiveType VARCHAR(20),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id) ON DELETE CASCADE
);

CREATE TABLE PREREQUISITE (
    Course_id VARCHAR(20),
    Prerequisite_id VARCHAR(20),
    PRIMARY KEY (Course_id, Prerequisite_id),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id),
    FOREIGN KEY (Prerequisite_id) REFERENCES COURSE(Course_id)
);

-- 4. COURSE CONTENT & SECTIONS

CREATE TABLE SECTION (
    Sec_id INT PRIMARY KEY,
    Sec_no VARCHAR(20),
    Year INT,
    Day VARCHAR(15),
    Room_Bldg_no VARCHAR(50),
    Room_Room_no VARCHAR(50),
    Course_id VARCHAR(20) NOT NULL,
    Teacher_id INT NOT NULL,
    Assistant_id INT,
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id),
    FOREIGN KEY (Teacher_id) REFERENCES INSTRUCTOR(Instructor_ID),
    FOREIGN KEY (Assistant_id) REFERENCES INSTRUCTOR(Instructor_ID)
);

CREATE TABLE CONTENT (
    Course_id VARCHAR(20) NOT NULL,
    Content_id INT NOT NULL,
    Week INT,
    Subject TEXT,
```

```
    Document VARCHAR(255),
    Practice VARCHAR(100),
    Method VARCHAR(100),
    Preparatory VARCHAR(255),
    PRIMARY KEY (Course_id, Content_id),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id) ON DELETE CASCADE
);

CREATE TABLE TEXTBOOK (
    Course_id VARCHAR(20),
    Textbook_Name VARCHAR(200),
    Edition VARCHAR(100),
    PRIMARY KEY (Course_id, Textbook_Name),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id) ON DELETE CASCADE
);

CREATE TABLE OUTCOME (
    Course_id VARCHAR(20),
    Outcome_Desc TEXT,
    Assesment_Method TEXT,
    Teaching_Method TEXT,
    PRIMARY KEY (Course_id, Outcome_Desc),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id) ON DELETE CASCADE
);

-- 5. ASSESSMENT SYSTEM

CREATE TABLE ASSESSMENT (
    Course_id VARCHAR(20),
    Assessment_id INT,
    Term_Percent DECIMAL(5,2),
    End_Term_Percent DECIMAL(5,2),
    PRIMARY KEY (Course_id, Assessment_id),
    FOREIGN KEY (Course_id) REFERENCES COURSE(Course_id) ON DELETE CASCADE,
    CONSTRAINT chk_total_100 CHECK ((Term_Percent + End_Term_Percent) = 100)
);

CREATE TABLE COMPONENT (
    Course_id VARCHAR(20),
    Assessment_id INT,
    Comp_Name VARCHAR(100),
    Type VARCHAR(50),
    Weight DECIMAL(5,2),
    Quantity INT,
    PRIMARY KEY (Course_id, Assessment_id, Comp_Name),
    FOREIGN KEY (Course_id, Assessment_id) REFERENCES ASSESSMENT(Course_id,
Assessment_id) ON DELETE CASCADE,
    CONSTRAINT check_type_valid CHECK (Type IN ('Term', 'End_Term'))
);

-- 6. STUDENT MOBILITY (EXCHANGE)

CREATE TABLE EXCHANGE_STUDENT (
    S_no INT PRIMARY KEY,
    Sname_Fname  VARCHAR(50),
    Sname_Mname  VARCHAR(50),
    Sname_Lname  VARCHAR(50),
    Major VARCHAR(100),
    Addr VARCHAR(200),
```

```
    Phone VARCHAR(20),
    DOB DATE,
    Dep_id INT NOT NULL,
    Since_date DATE,
    FOREIGN KEY (Dep_id) REFERENCES DEPARTMENT(Dep_id)
);

CREATE TABLE E_CANDIDATE (
    Dep_id INT,
    S_no INT,
    App_id INT,
    App_date DATE,
    Status VARCHAR(50),
    PRIMARY KEY (Dep_id, S_no, App_id),
    FOREIGN KEY (Dep_id) REFERENCES DEPARTMENT(Dep_id),
    FOREIGN KEY (S_no) REFERENCES EXCHANGE_STUDENT(S_no),
    CONSTRAINT chk_status_valid_app CHECK (Status IN ('Approved', 'Rejected', 'Pending',
'Conditional'))
);

CREATE TABLE EQUIVALENCY (
    Host_Course_id VARCHAR(20),
    Home_Course_id VARCHAR(20),
    Language_match BOOLEAN,
    ECTS_diff INT,
    Status VARCHAR(50),
    PRIMARY KEY (Host_Course_id, Home_Course_id),
    FOREIGN KEY (Host_Course_id) REFERENCES COURSE(Course_id),
    FOREIGN KEY (Home_Course_id) REFERENCES COURSE(Course_id),
    CONSTRAINT chk_status_valid_eq CHECK (Status IN ('NOT_EQUAL', 'EQUAL'))
);
```

5.2) Implementation Notes

- **Data Integrity:** Constraints such as `CHECK`, `NOT NULL`, and `UNIQUE` were used. Business rules (e.g., ensuring total assessment weights equal 100) are enforced directly at the database level.
- **Cascading Deletes:** The `ON DELETE CASCADE` rule is applied to strong-weak entity relationships and subtypes to ensure that deleting a parent record (e.g., a Course) automatically removes its associated details (e.g., Outcomes, Textbooks, Assessments).

5.3) Check Constraints:

COURSE (`chk_ects_valid`):

- **Logic:** The ECTS value must be higher than 0.
- **Meaning:** This ensures that a course has a valid credit value. It prevents errors like negative credits.

ASSESSMENT (`chk_total_100`):

- **Logic:** The sum of `Term_Percent` and `End_Term_Percent` must equal exactly 100.
- **Meaning:** This guarantees that the grading scheme is mathematically valid. The semester work and the final exam must combine to make a total of 100%.

COMPONENT (`check_type_valid`):

- **Logic:** The text entered must be either `'Term'` or `'End_Term'`.
- **Meaning:** This restricts data entry to these two specific options. It prevents inconsistency (e.g., preventing someone from typing "Finals" instead of "End_Term").

E_CANDIDATE (`chk_status_valid`):

- **Logic:** The status must be one of four specific words: `'Approved'`, `'Rejected'`, `'Pending'`, or `'Conditional'`.
- **Meaning:** This standardizes the application process. An administrator cannot invent a new status (like "Waiting") that the system doesn't recognize.

EQUIVALENCY (`chk_status_valid`):

- **Logic:** The status must be either `'EQUAL'` or `'NOT_EQUAL'`.
- **Meaning:** This simplifies course matching into a strictly binary decision: either the courses match, or they do not.

5.4) Triggers

1. Prevent Home Country Application Trigger

This trigger enforces the Erasmus+ mobility rule stating that an exchange student cannot apply to a department located in their home country.

Before inserting a new candidate application, the trigger compares the student's home department country with the target department country. If both countries are the same, the transaction is aborted.

```
-- 1) Prevent Home Country Application
CREATE OR REPLACE FUNCTION check_home_department_application()
RETURNS TRIGGER AS $$
DECLARE
    student_home_country VARCHAR(20);
    target_dept_country VARCHAR(20);
BEGIN
    -- Step 1: Retrieve the home country of the applicant
    -- Join EXCHANGE_STUDENT with DEPARTMENT to find the origin country
    SELECT D.Country INTO student_home_country
    FROM EXCHANGE_STUDENT S
    JOIN DEPARTMENT D ON S.Dep_id = D.Dep_id
    WHERE S.S_no = NEW.S_no;

    -- Step 2: Retrieve the country of the target department
    SELECT Country INTO target_dept_country
    FROM DEPARTMENT
    WHERE Dep_id = NEW.Dep_id;

    -- Step 3: Validate the mobility rule
    -- If countries match, raise an exception and abort the transaction
    IF student_home_country = target_dept_country THEN
        RAISE EXCEPTION 'Violation of Mobility Rule: Student cannot apply to a department in their
home country (%).', student_home_country;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

/* Trigger Definition */
DROP TRIGGER IF EXISTS trg_prevent_home_application ON E_CANDIDATE;

CREATE TRIGGER trg_prevent_home_application
BEFORE INSERT ON E_CANDIDATE
FOR EACH ROW
EXECUTE FUNCTION check_home_department_application();
```

2. Dynamic Curriculum ECTS Calculation Trigger

This trigger ensures that the total ECTS value of a curriculum is always consistent with the sum of the ECTS values of its courses.

Whenever a course is inserted, updated, or deleted, the total ECTS of the related curriculum is recalculated automatically.

```
-- 2)Dynamic ECTS Calculation

CREATE OR REPLACE FUNCTION auto_update_curriculum_ects()
RETURNS TRIGGER AS $$
BEGIN
    -- Case 1: Handle INSERT or UPDATE actions
    -- Recalculate total ECTS for the referenced curriculum
    IF (TG_OP = 'INSERT' OR TG_OP = 'UPDATE') THEN
        UPDATE CURRICULUM
        SET Total_ects = (SELECT COALESCE(SUM(ECTS), 0) FROM COURSE WHERE Curr_id =
NEW.Curr_id)
        WHERE Curr_id = NEW.Curr_id;
    END IF;

    -- Case 2: Handle DELETE or UPDATE actions
    -- Recalculate total ECTS for the old curriculum (in case of a transfer)
    IF (TG_OP = 'DELETE' OR TG_OP = 'UPDATE') THEN
        UPDATE CURRICULUM
        SET Total_ects = (SELECT COALESCE(SUM(ECTS), 0) FROM COURSE WHERE Curr_id =
OLD.Curr_id)
        WHERE Curr_id = OLD.Curr_id;
    END IF;

    RETURN NULL; -- Return NULL for AFTER triggers
END;
$$ LANGUAGE plpgsql;

/* Trigger Definition */
CREATE TRIGGER trg_maintain_curriculum_ects
AFTER INSERT OR UPDATE OR DELETE ON COURSE
FOR EACH ROW
EXECUTE FUNCTION auto_update_curriculum_ects();
```

3. Component Weight Validation Trigger

This trigger guarantees that the total weight of assessment components (e.g., midterm, final, project) for a specific course and assessment type does not exceed 100%.

If the total weight exceeds this limit after an insert or update, the operation is rejected.

```
-- 3) Component Weight Validation

CREATE OR REPLACE FUNCTION check_component_weight_limit()
RETURNS TRIGGER AS $$
DECLARE
    current_total DECIMAL(5,2);
BEGIN
    -- Calculate the sum of weights for the specific Course, Assessment, and Type
    -- Note: This is an AFTER trigger, so the new row is already included in the sum
    SELECT COALESCE(SUM(Weight), 0) INTO current_total
    FROM COMPONENT
    WHERE Course_id = NEW.Course_id
      AND Assessment_id = NEW.Assessment_id
      AND Type = NEW.Type;

    -- Check if the total exceeds 100
    IF current_total > 100 THEN
        RAISE EXCEPTION 'Total Weight for % components cannot exceed 100. Current total is %.',
NEW.Type, current_total;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

/* Trigger Definition */
CREATE TRIGGER trg_weight_limit
AFTER INSERT OR UPDATE ON COMPONENT
FOR EACH ROW
EXECUTE FUNCTION check_component_weight_limit();
```

4. Automatic Equivalency Detail Calculation Trigger

This trigger automatically calculates derived attributes in the EQUIVALENCY table.

It determines whether the languages of the host and home courses match and computes the absolute difference between their ECTS values.

```
CREATE OR REPLACE FUNCTION calculate_equivalency_details()
RETURNS TRIGGER AS $$
DECLARE
```

```
    host_ects INT;
    host_lang VARCHAR;
    home_ects INT;
    home_lang VARCHAR;
BEGIN
    -- 1. Fetch details of the Host Course
    SELECT ECTS, Language INTO host_ects, host_lang
    FROM COURSE
    WHERE Course_id = NEW.Host_course_id;

    -- 2. Fetch details of the Home Course
    SELECT ECTS, Language INTO home_ects, home_lang
    FROM COURSE
    WHERE Course_id = NEW.Home_course_id;

    -- 3. Determine Language Match
    IF host_lang = home_lang THEN
        NEW.Language_match := TRUE;
    ELSE
        NEW.Language_match := FALSE;
    END IF;

    -- 4. Calculate Absolute ECTS Difference
    NEW.ECTS_diff := ABS(host_ects - home_ects);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

/* Trigger Definition */
CREATE TRIGGER trg_set_equivalency_details
BEFORE INSERT OR UPDATE ON EQUIVALENCY
FOR EACH ROW
EXECUTE FUNCTION calculate_equivalency_details();
```

5. Self-Reference Prevention Triggers

These triggers protect data integrity by preventing invalid self-referencing relationships.

5.1 Prevent Self-Equivalency

A course cannot be marked as equivalent to itself.

```
-- 5) Self-Reference Prevention

/* Function 1: Prevent Self-Equivalency */
CREATE OR REPLACE FUNCTION prevent_self_equivalency()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Host_Course_id = NEW.Home_Course_id THEN
        RAISE EXCEPTION 'Data Integrity Error: A course cannot be equivalent to itself.';
```

```
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_no_self_equivalency
BEFORE INSERT OR UPDATE ON EQUIVALENCY
FOR EACH ROW
EXECUTE FUNCTION prevent_self_equivalency();
```

5.2 Prevent Self-Prerequisite

A course cannot be defined as its own prerequisite.

```
/* Function 2: Prevent Self-Prerequisite */
CREATE OR REPLACE FUNCTION check_self_prerequisite_func()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Course_id = NEW.Prerequisite_id THEN
        RAISE EXCEPTION 'Data Integrity Error: A course (%s) cannot be its own prerequisite.',
NEW.Course_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_self_prerequisite
BEFORE INSERT OR UPDATE ON PREREQUISITE
FOR EACH ROW
EXECUTE FUNCTION check_self_prerequisite_func();
```

6) CONCLUSION

In this term project, we designed and developed a relational database system for the Erasmus+ Exchange Programme. Our goal was to manage student applications and course equivalencies between different universities efficiently.

The development process consisted of three main stages:

- **Design and Modeling:** First, we analyzed the requirements of Ege University, Muğla Sıtkı Koçman University, and Sakarya University. We merged their different structures into a single Integrated EER Model. After finalizing the diagram, we converted this model into the Logical Model (Relational Schema). This step was crucial to correctly define the tables, primary keys, and relationships before writing the code.
- **SQL Implementation:** We implemented the physical database using SQL. To ensure data accuracy, we used Triggers and Check Constraints. These automated rules help the system to:
    - Prevent students from applying to a university in their home country.
    - Automatically calculate the total ECTS credits of a curriculum.
    - Ensure that exam and assignment weights always sum up to 100.
- **Testing:** We verified the system using various SQL queries. The results showed that the database successfully handles complex tasks like listing course schedules, tracking application statuses, and comparing course contents.