Task 2 — Generating Design of Other Traditional Database Models

🎯 Objective:

To convert an abstract data model (University Management System) into a Hierarchical/Network database model, then extend it using inheritance (generalization/specialization) concepts and implement it in SQL.

2.a Identify the Specificity of Each Relationship and Form Surplus Relations

Entities and Relationships

Relationship	Type	Description
Student is-a Person	IS-A	Specialization of Person
Professor is-a Person	IS-A	Specialization of Person
Professor teaches Course	HAS-A	Each course is taught by one professor
Student enrolls in Course	Many-to-Many	Students can take multiple courses
Course belongs to Department	HAS-A	A department offers many courses

Surplus Relationships

Some relationships can be derived (do not need explicit storage):

Derived Relationship	Derived From	Status
Student → Department	Student → Course + Course → Department	Surplus (derived)
Professor → Department	Professor teaches Course in Department	Surplus (derived)

✅ Therefore, only core relationships are stored explicitly.

2.b Check IS-A / HAS-A Hierarchy and Perform Generalization / Specialization

Generalization

Department HAS-A Professor

Course HAS-A Professor

Course HAS-A Department

Student ENROLLS-IN Course

2.c Find Domain of Attributes and Apply Check Constraints

Attribute	Domain	SQL Check Constraint
age	INTEGER	CHECK (age BETWEEN 18 AND 100)
gender	ENUM ('Male','Female')	CHECK (gender IN ('Male','Female'))
gpa	DECIMAL(3,2)	CHECK (gpa BETWEEN 0.00 AND 4.00)
course_credits	INTEGER	CHECK (course_credits BETWEEN 1 AND 6)

2.d Rename the Relations

Old Name	New Name
Person	tbl_persons
Student	tbl_students
Professor	tbl_professors
Course	tbl_courses
Department	tbl_departments

Syntax Example:

```
RENAME TABLE Student TO tbl_students;
```

2.e Perform SQL Relations Using DDL and DCL

🧱 Data Definition Language (DDL)

Superclass

```
CREATE TABLE tbl_persons (
   person_id INT PRIMARY KEY,
   name VARCHAR(100),
   age INT CHECK (age BETWEEN 18 AND 100),
   gender VARCHAR(10) CHECK (gender IN ('Male', 'Female'))
);
```

Specialization: Students

```
CREATE TABLE tbl_students (
   student_id INT PRIMARY KEY,
   person_id INT,
   gpa DECIMAL(3,2) CHECK (gpa BETWEEN 0.00 AND 4.00),
   FOREIGN KEY (person_id) REFERENCES tbl_persons(person_id)
);
```

Combining similar entities into a common superclass:

```
Person
  ├── Student
  └── Professor
```

Specialization

Splitting based on unique attributes:

Subclass	Unique Attributes
Student	roll_no, gpa
Professor	emp_id, department_id

HAS-A RelationshipsSpecialization: Professors

```
CREATE TABLE tbl_professors (
    professor_id INT PRIMARY KEY,
    person_id INT,
    department_id INT,
    FOREIGN KEY (person_id) REFERENCES tbl_persons(person_id)
);
```

Departments

```
CREATE TABLE tbl_departments (
    department_id INT PRIMARY KEY,
    name VARCHAR(100)
);
```

Courses

```
CREATE TABLE tbl_courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    course_credits INT CHECK (course_credits BETWEEN 1 AND 6),
    department_id INT,
    professor_id INT,
    FOREIGN KEY (department_id) REFERENCES tbl_departments(department_id),
    FOREIGN KEY (professor_id) REFERENCES tbl_professors(professor_id)
);
```

Enrollments

```
CREATE TABLE tbl_enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
```

```sql
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES tbl_students(student_id),
    FOREIGN KEY (course_id) REFERENCES tbl_courses(course_id)
);
```

🧩 Example INSERT Queries

```sql
-- Persons
INSERT INTO tbl_persons VALUES (1, 'Alice Johnson', 22, 'Female');
INSERT INTO tbl_persons VALUES (2, 'Dr. Smith', 45, 'Male');

-- Departments
INSERT INTO tbl_departments VALUES (101, 'Computer Science');
INSERT INTO tbl_departments VALUES (102, 'Mathematics');

-- Professors
INSERT INTO tbl_professors VALUES (201, 2, 101);

-- Students
INSERT INTO tbl_students VALUES (301, 1, 3.80);

-- Courses
INSERT INTO tbl_courses VALUES (401, 'Database Systems', 4, 101, 201);
INSERT INTO tbl_courses VALUES (402, 'Algorithms', 3, 101, 201);

-- Enrollments
INSERT INTO tbl_enrollments VALUES (501, 301, 401);
INSERT INTO tbl_enrollments VALUES (502, 301, 402);
```

🔐 Data Control Language (DCL)

Granting Privileges

```
GRANT SELECT, INSERT, UPDATE ON tbl_students TO some_user;

GRANT SELECT ON tbl_courses TO some_user;
```

Revoking Privileges

```
REVOKE UPDATE ON tbl_students FROM some_user;
```

📊 Final Model Representation

Hierarchical Model

```
tbl_persons
 ├── tbl_students
└── tbl_professors
    └── tbl_courses
        └── tbl_enrollments
```

Network Model

Relationship	Type
Students ↔ Courses	Many-to-Many
Professors → Courses	One-to-Many
Courses → Departments	Many-to-One
Departments ↔ Professors	One-to-Many

✅ Final Summary:

We started with an abstract University data model.

Applied inheritance (generalization/specialization) using tbl_persons as superclass.

Defined domains and constraints for data integrity.

Used DDL for table creation and DCL for access control.

Represented both Hierarchical and Network relationships