YrkesCo Database Modeling

A structured relational design from needs to implementation

Project Overview

- Goal: Replace Excel tracking with a normalized relational database.
- Scope: Manage schools, programs, courses, students, teachers, and roles.
- Focus: Scalability, security, and data integrity.

Method: From Requirements to Model

We began by analyzing the business needs of YrkesCo:

- Multiple campuses with different schools
- Teachers can be consultants or employees
- Students need to enroll in classes and standalone courses
- Sensitive personal data must be secured
- Courses are reused across programs
- From these needs, we derived entities and relationships using best practices from data modeling. We then moved from abstract thinking to technical implementation through three major steps: conceptual design, logical design, and physical design.

Conceptul Model

The conceptual model represents the foundation of the database system. It is a high-level, business-oriented abstraction of YrkesCo's operations and identifies the main entities involved in the system without including database-specific details such as data types or keys.

Its purpose is to capture **real-world objects** and **relationships** from a business perspective and prepare the structure for later technical development stages.

This model is essential to ensure that all stakeholders—both technical and non-technical—can agree on the system's intended functionality before moving forward to more detailed modeling.

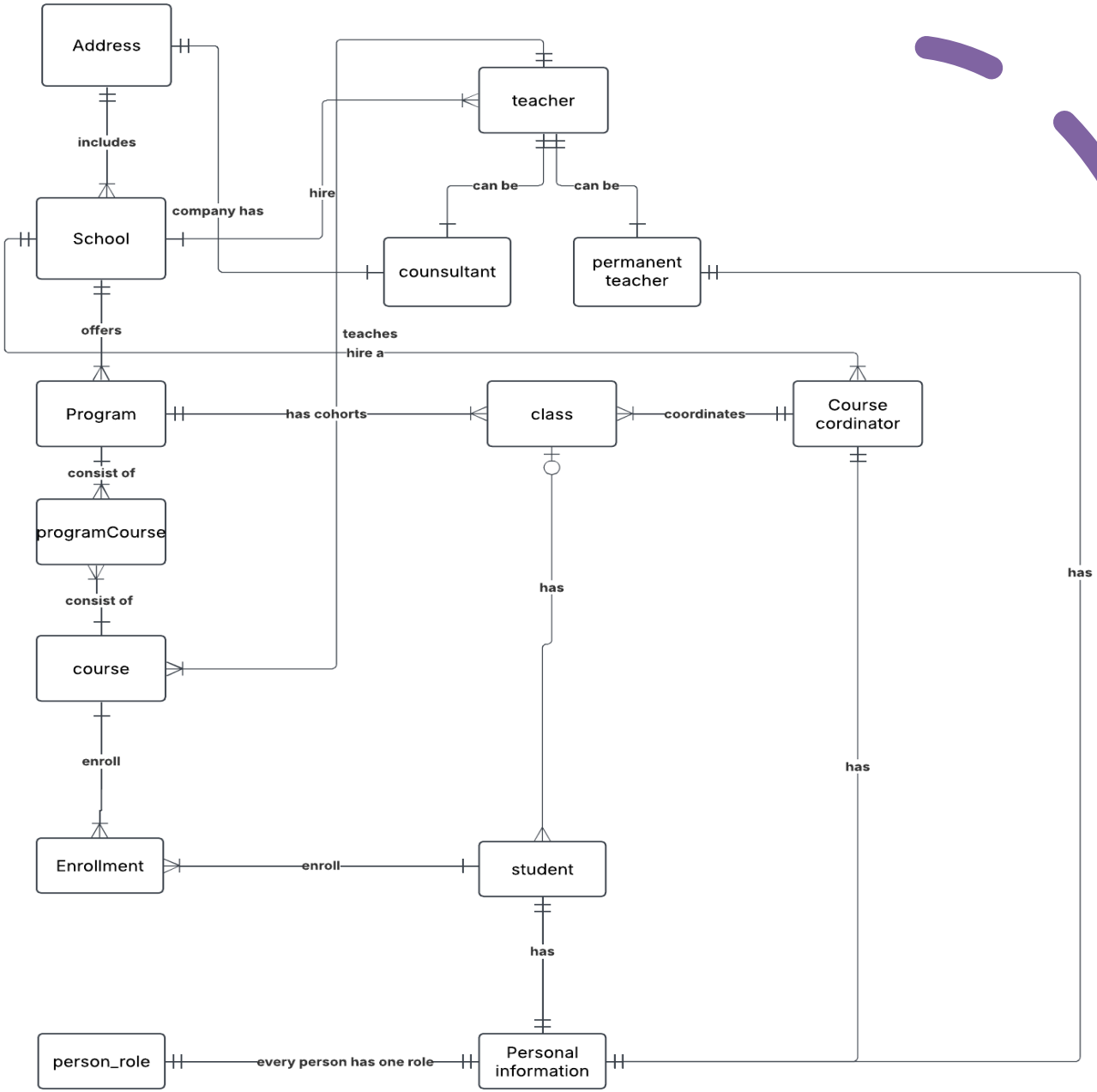
The conceptual model outlines the core entities and their relationships at a high level. It answers: **"What are the key actors in the system and how do they interact?"**

Key components:

- **School** offers **Programs**
- Each **Program** consists of multiple **Courses**
- **Classes** are instances of Programs; each is coordinated by one **Course Coordinator**
- **Students** are enrolled in Classes and Courses
- **Teachers** can be **Consultants** or **Permanent Teachers**
- **Address** is abstracted and reused across entities

This model avoids technical specifics and focuses on the business logic and process.

ERD


Address
includes
School
company has
hire
teacher
can be
can be
counsultant
permanent teacher
offers
teaches
hire a
Program
has cohorts
class
coordinates
Course cordinator
consist of
programCourse
consist of
course
has
has
enroll
enroll
Enrollment
enroll
student
has
person_role
every person has one role
Personal information

Logical Model

We transformed the conceptual model into a logical schema with attributes, constraints, and unique identifiers (primary and foreign keys). Each entity now includes necessary fields, data types, and relationships with other entities.

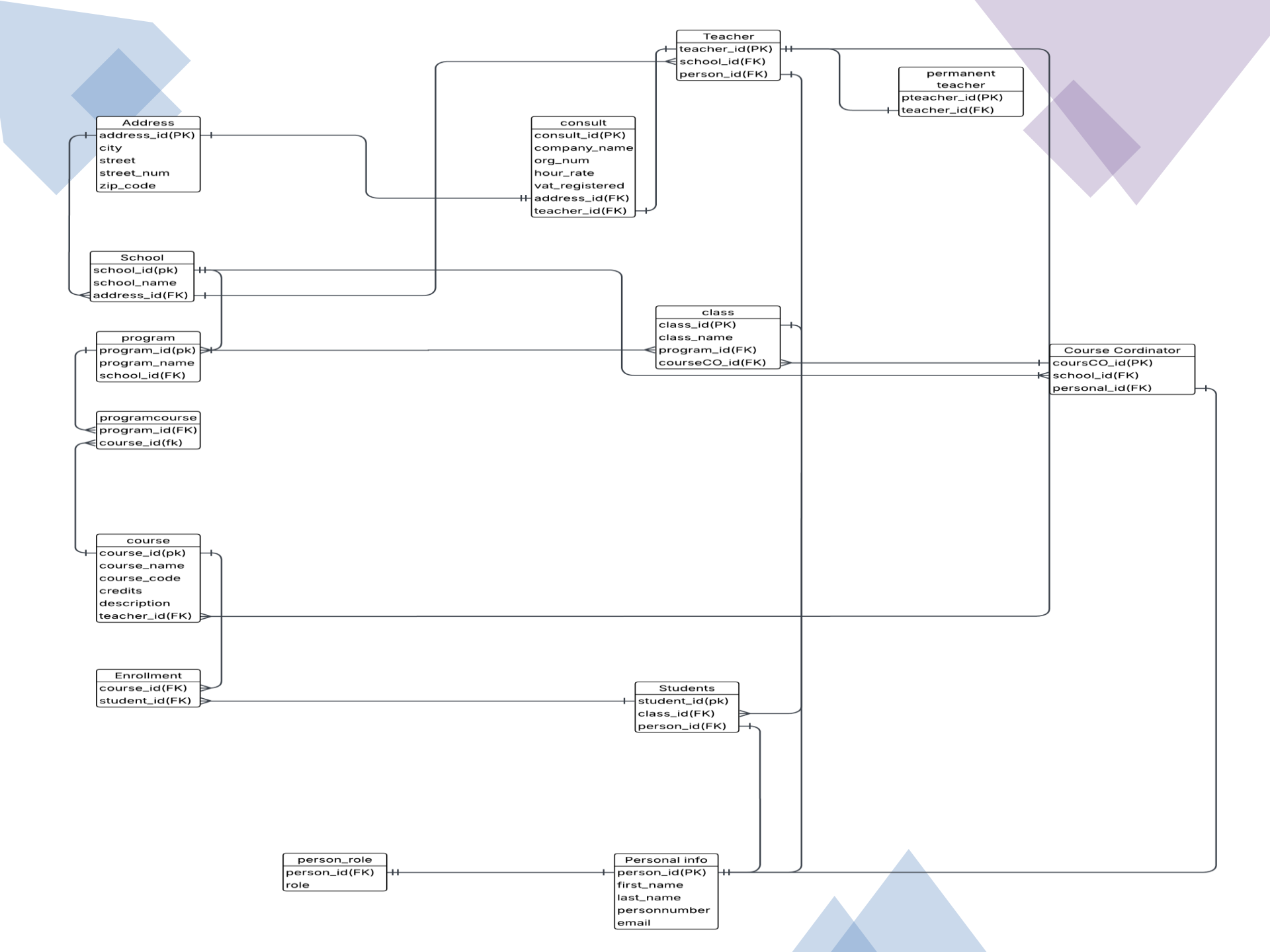
Key decisions:

- personal_info table holds sensitive personal data; reused via person_id in related tables.
- person_role allows each individual to hold one or more roles (teacher, student, coordinator).
- Subtypes of teacher include permanent_teacher and consultant, using foreign keys to generalize the concept of teacher.

Why Bridge Tables? Bridge tables like programcourse and enrollment were introduced to handle many-to-many relationships:

- A program consists of many courses, and courses can appear in multiple programs.
- A student can enroll in multiple courses, and each course can be taken by many students.

Using bridge tables supports normalization, flexibility in querying, and avoids duplication.


Teacher
teacher_id(PK)
school_id(FK)
person_id(FK)
permanent teacher
pteacher_id(PK)
teacher_id(FK)
Address
address_id(PK)
city
street
street_num
zip_code
consult
consult_id(PK)
company_name
org_num
hour_rate
vat_registered
address_id(FK)
teacher_id(FK)
School
school_id(pk)
school_name
address_id(FK)
class
class_id(PK)
class_name
program_id(FK)
courseCO_id(FK)
program
program_id(pk)
program_name
school_id(FK)
Course Cordinator
coursCO_id(PK)
school_id(FK)
personal_id(FK)
programcourse
program_id(FK)
course_id(fk)
course
course_id(pk)
course_name
course_code
credits
description
teacher_id(FK)
Enrollment
course_id(FK)
student_id(FK)
Students
student_id(pk)
class_id(FK)
person_id(FK)
person_role
person_id(FK)
role
Personal info
person_id(PK)
first_name
last_name
personnumber
email

Relationship Concepts and Glossary

- **Primary Key (PK):** A unique identifier (e.g., person_id)
- **Foreign Key (FK):** A reference to a PK in another table (e.g., school_id in program)
- **One-to-Many (1:N):** One school has many programs
- **Many-to-Many (M:N):** Courses are shared among programs; students can take many courses
- **One-to-One (1:1):** One person_id per student or coordinator via unique constraint
- **Generalization:** teacher is generalized into two specializations — consultant and employee
- **Entity:** A business concept (e.g., student, class)
- **Attribute:** A data point describing an entity (e.g., first_name, email)

This step ensures logical correctness and readiness for database implementation.

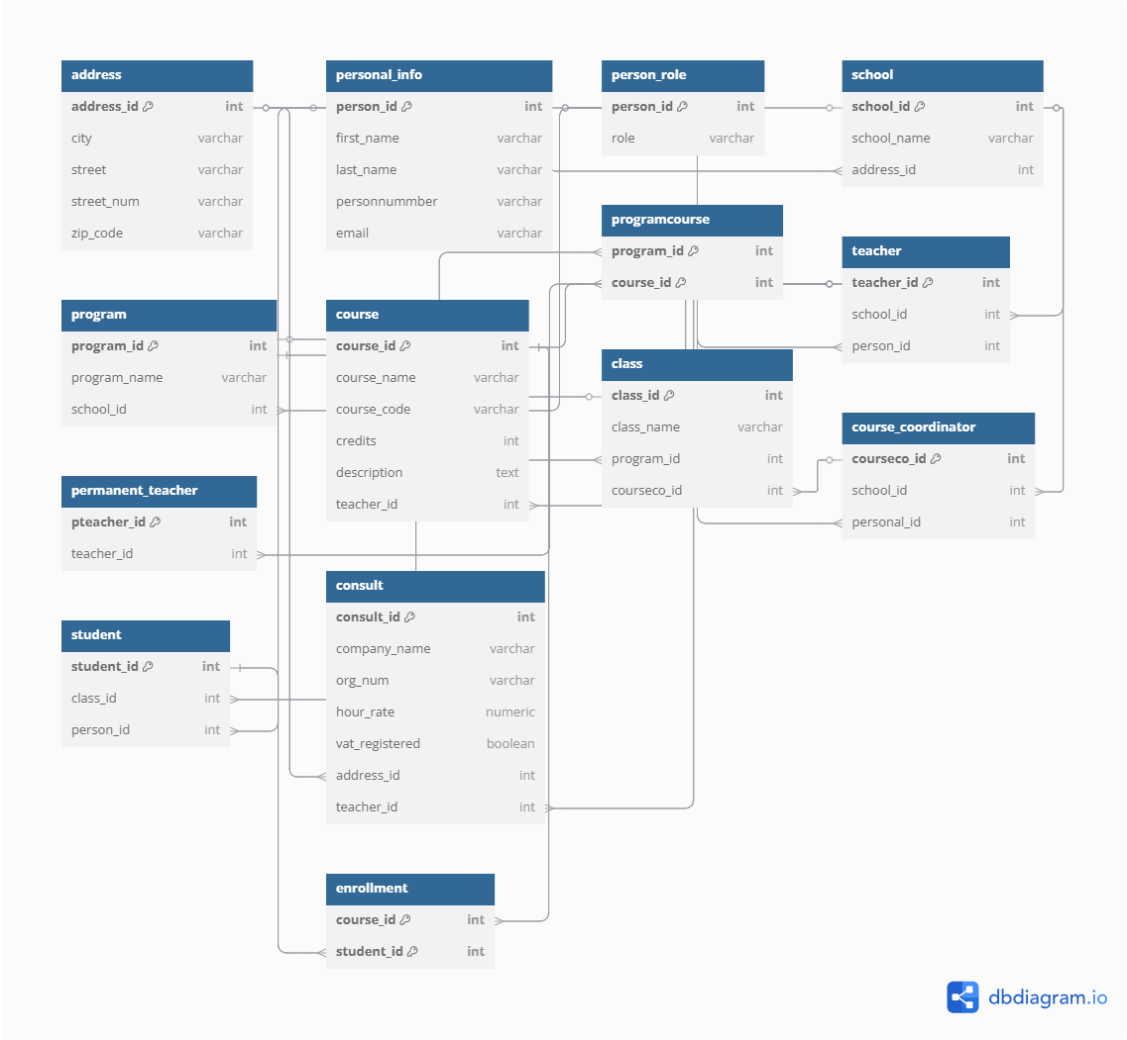
Physical Model

The physical model defines how we actually implement this structure in PostgreSQL.

Key features:

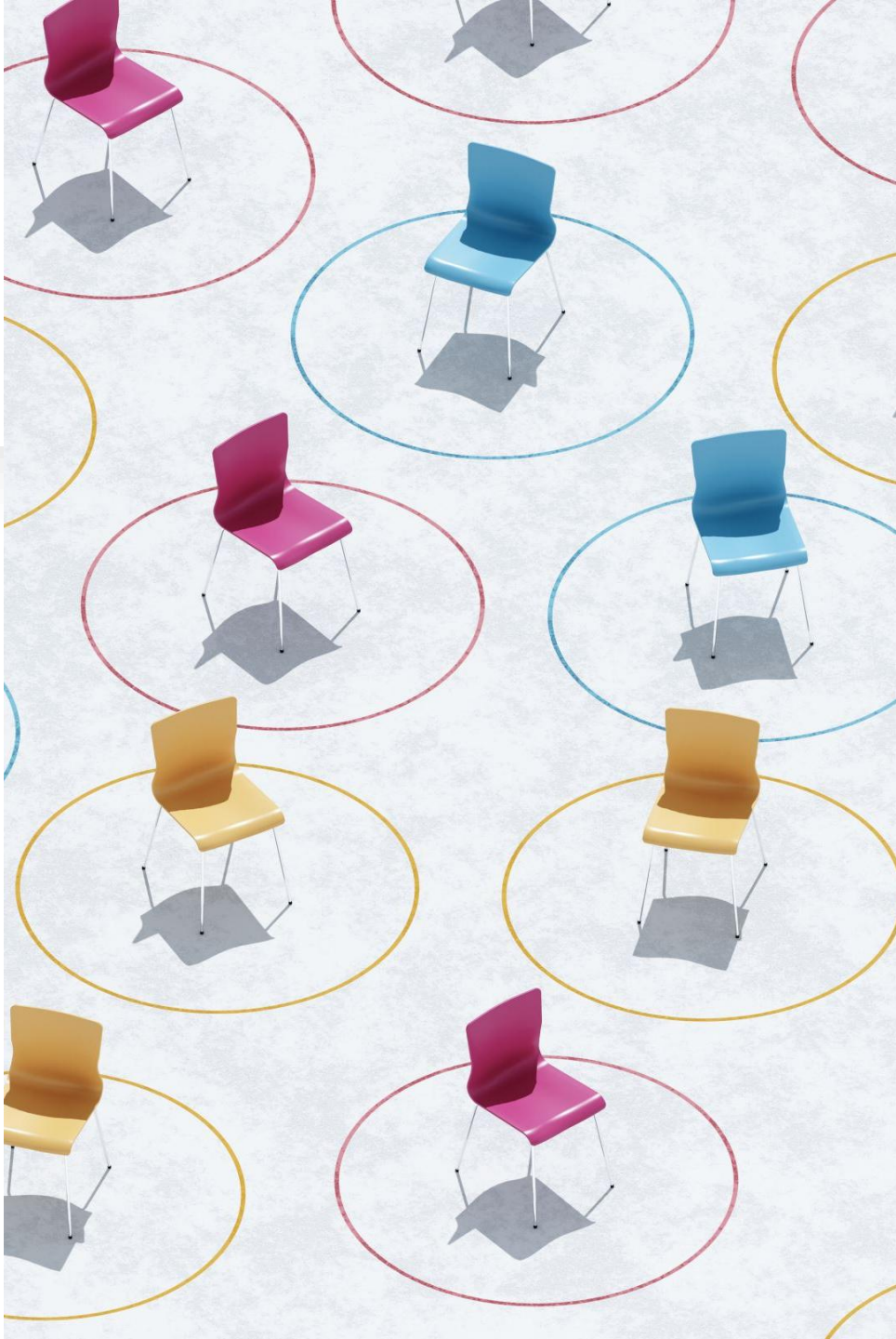
- All primary keys are integers with auto-increment
- Foreign keys maintain referential integrity
- Composite primary keys in enrollment and programcourse
- Use of unique constraints to enforce 1:1 relations (e.g., student ↔ personal_info)
- Standard naming conventions and data types used (e.g., varchar, int, boolean)

The physical model was exported as a .dbml file for use in dbdiagram.io and PostgreSQL. It supports automated visualization and system implementation.


address
address_id int
city varchar
street varchar
street_num varchar
zip_code varchar
personal_info
person_id int
first_name varchar
last_name varchar
personnummber varchar
email varchar
person_role
person_id int
role varchar
school
school_id int
school_name varchar
address_id int
programcourse
program_id int
course_id int
teacher
teacher_id int
school_id int
person_id int
program
program_id int
program_name varchar
school_id int
course
course_id int
course_name varchar
course_code varchar
credits int
description text
teacher_id int
class
class_id int
class_name varchar
program_id int
courseco_id int
course_coordinator
courseco_id int
school_id int
personal_id int
permanent_teacher
pteacher_id int
teacher_id int
consult
consult_id int
company_name varchar
org_num varchar
hour_rate numeric
vat_registered boolean
address_id int
teacher_id int
student
student_id int
class_id int
person_id int
enrollment
course_id int
student_id int
dbdiagram.io

Why Bridge Tables?

- Many-to-many relationships (e.g., course <-> student)
- Avoids redundancy
- Ensures scalable and clean design



Normalization & 3NF

1NF: All fields are atomic

2NF: All attributes depend on the whole key

3NF: No transitive dependencies

Example: student.email stored only in personal_info

Summary

- Fully normalized schema (3NF)
- Reflects real business structure
- Ready for PostgreSQL implementation
- Supports future growth