Datamodellering för YrkesCo

Ett datadrivet utbildningssytem

Robin Sundman Nilsson — DE24

Projektets syfte

Skapa en strukturerad och skalbar databaslösning åt YrkesCo som kan hantera:

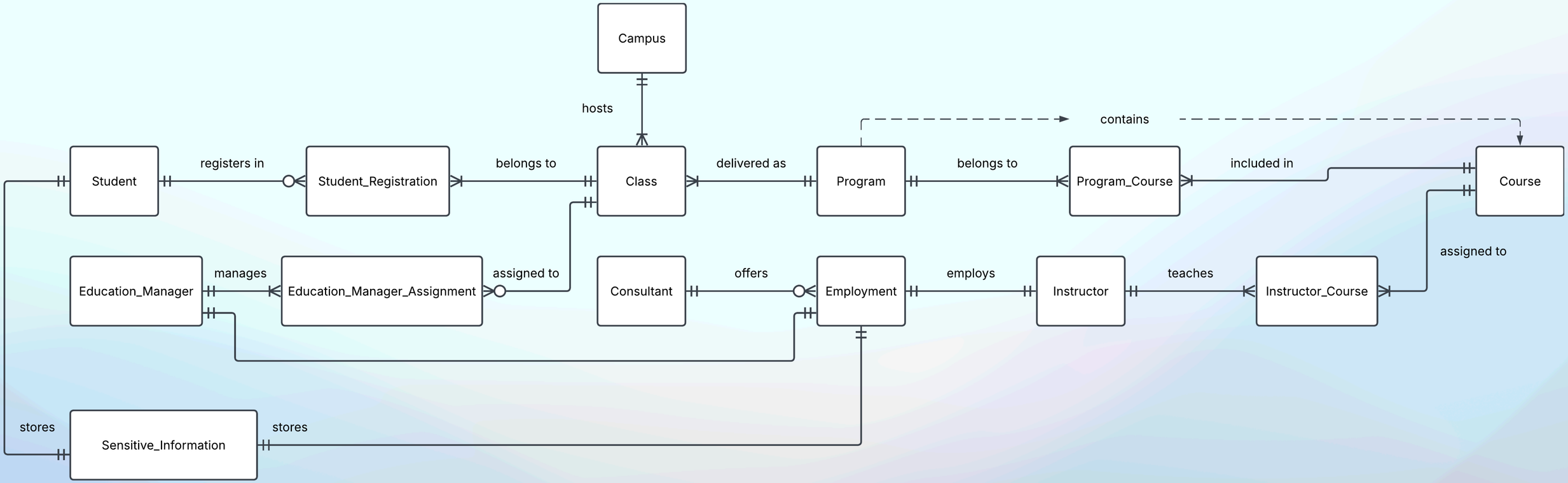
- Planering av utbildningar
- Administration av studenter, personal, program, klasser & campus
- Underlag för analys och framtida rapportering

Kort om processen

- Konceptuell Modell — vilka objekt finns?
- Logisk Modell — hur hänger de ihop?
- Fysisk Modell — hur ser det ut i databasen?
- Implementation & teknisk miljö— PostgreSQL + Docker
- Datainmatning — Fiktiv men realistisk data
- Test — SQL-queries & demo

Konceptuell modell

Affärsobjekten



Konceptuell modell

Övergripande koncept och struktur

Den konceptuella modellen är verksamhetsnära, syftet är att fånga affärsobjekt och verksamhetsregler.

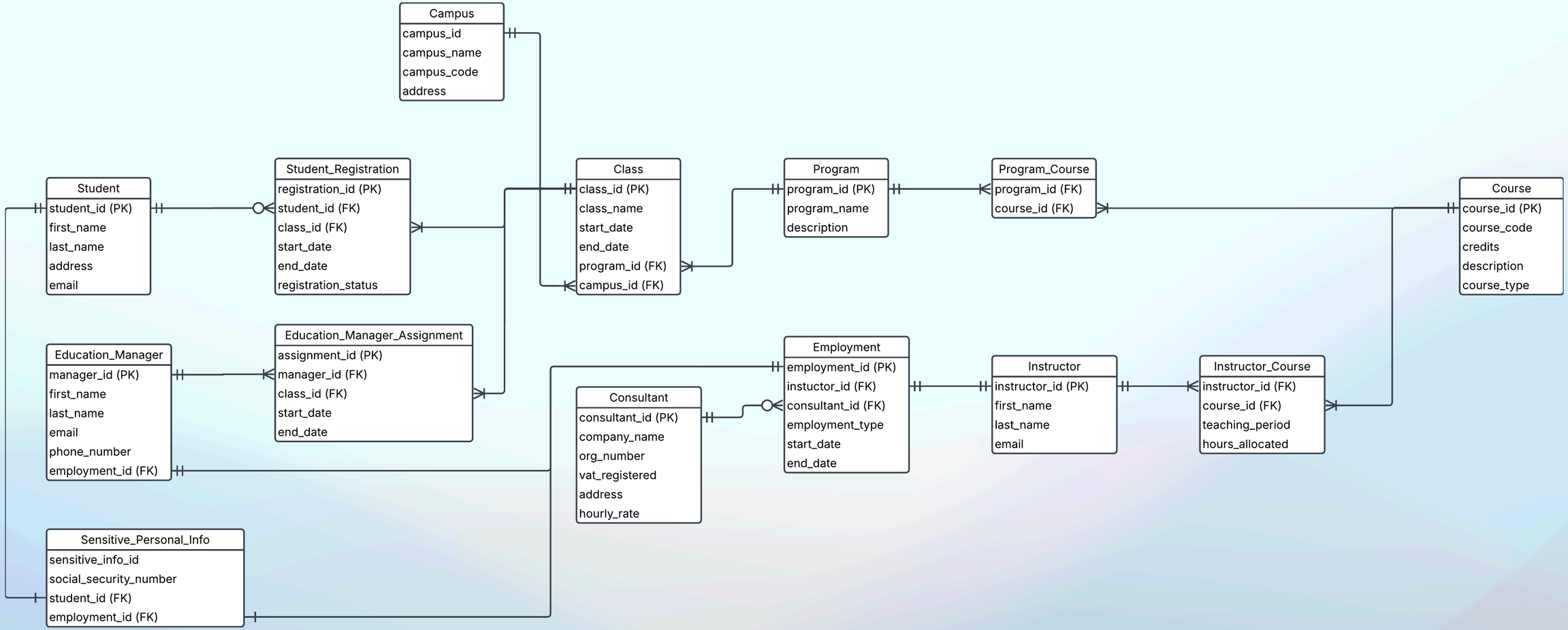
Fokus här låg på **vad** som är viktigt för YrkesCo, inte hur det sparas.

Några nyckelobjekt: Student, Instruktör, Utbildningsledare, Klass, Kurs & Program.

Målet är att förstå YrkesCo’s behov innan vi går vidare med den tekniska biten.

Logisk modell

Från koncept till struktur



Logisk modell

Struktur, nycklar och relationer

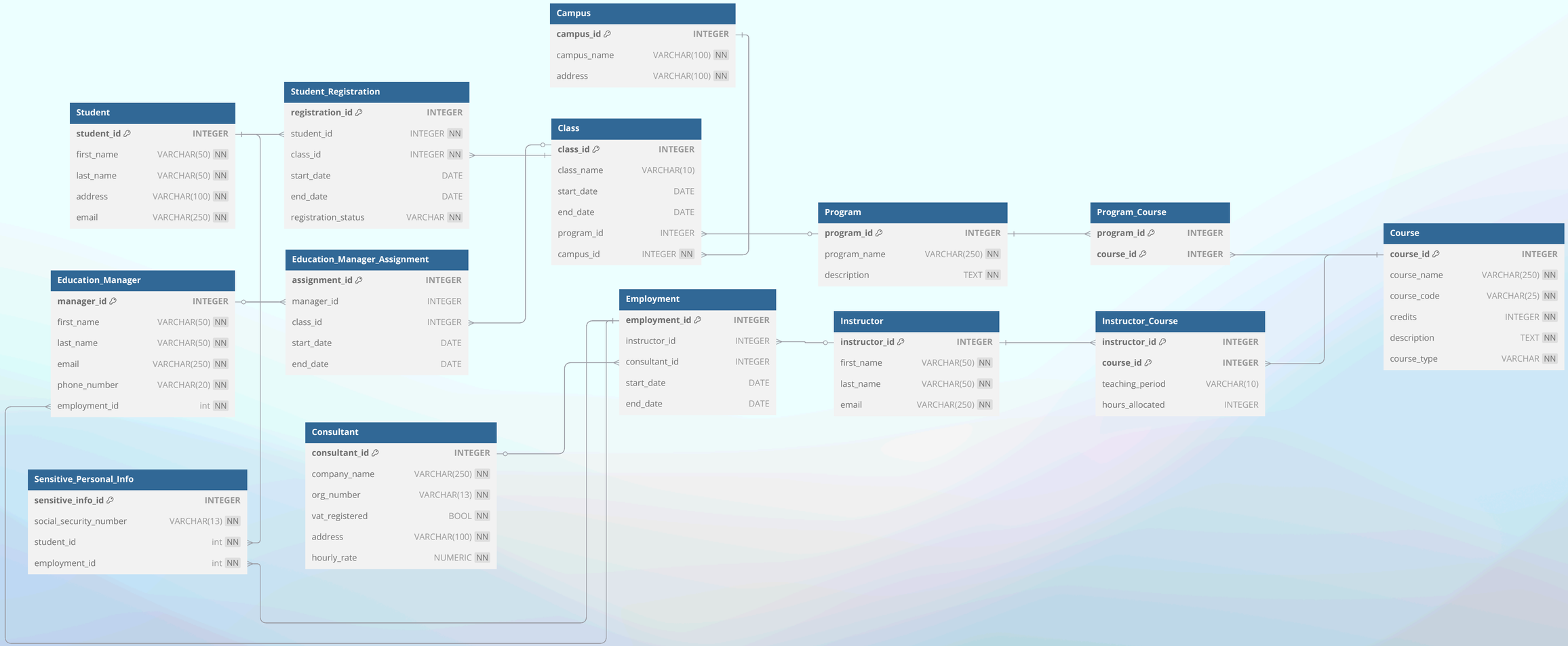
Här översätter vi den konceptuella modellen, skapar entiteter, definierar attribut (t.ex. first_name, program_name), primärnycklar och främmande nycklar.

I detta steg har jag följt principerna för **3NF** (tredje normalformen), vilket innebär att alla tabeller har en **entydig identitet** och att **all information lagras på sin mest logiska plats**. Inga kolumner ska bero på andra icke-nyckelattribut, utan endast direkt på primärnyckeln. Detta minskar risken för uppdateringsfel och inkonsekvent data.

T.ex: en **utbildningsledare** har sitt eget **ID** och är lagrad i en separat tabell – inte hårdkodad till klassen. På så sätt kan flera klasser kopplas till samma utbildningsledare utan att behöva duplicera namn eller kontaktuppgifter, vilket gör modellen både flexibel och skalbar.

Fysisk modell

Databasstrukturen i verkligheten



Fysisk modell

Tabeller och datatyper

I detta steg så gör vi den sista översättningen, entiteter blir tabeller och attribut har blivit tabell kolumner.

Vi definierar primär- och främmande nycklar samt datatyper för varje kolumn.

Vi refererar våra nycklar och sätter **not null** på de kolumner där det behövs.

Målet här var att skapa en robust, stabil och tydlig struktur inför vår implementation.

```
Table Student {
  student_id INTEGER [primary key]
  first_name VARCHAR(50) [not null]
  last_name VARCHAR(50) [not null]
  address VARCHAR(100) [not null]
  email VARCHAR(250) [not null]
}

Table Student_Registration {
  registration_id INTEGER [primary key]
  student_id INTEGER [not null, ref: > Student.student_id]
  class_id INTEGER [not null, ref: > Class.class_id]
  start_date DATE
  end_date DATE
  registration_status VARCHAR [not null] // e.g. "active", "completed", "dropped"
}

Table Class {
  class_id INTEGER [primary key]
  class_name VARCHAR(10)
  start_date DATE
  end_date DATE
  program_id INTEGER [ref: > Program.program_id]
  campus_id INTEGER [not null, ref: > Campus.campus_id]
}

Table Program {
  program_id INTEGER [primary key]
  program_name VARCHAR(250) [not null]
  description TEXT [not null]
}

Table Program_Course {
  program_id INTEGER [primary key, ref: > Program.program_id]
  course_id INTEGER [primary key, ref: > Course.course_id]
}
```

Implementation

PostgreSQL

I detta steg så har jag översatt vår fysiska modell från DBML (Database Markup Language) till SQL kod för att användas i PostgreSQL.

Vi skapar ett eget **SCHEMA** för YrkesCo och ser till att alla tabeller hamnar där.

Här använder jag **SERIAL** för autoinkrementerande ID, **VARCHAR, DATE, INTEGER** som datatyper, **PRIMARY KEY**, **FOREIGN KEY** samt **NOT NULL** för dataintegritet.

```sql
CREATE SCHEMA IF NOT EXISTS yrkesco;
SET search_path TO yrkesco;

CREATE TABLE Campus (
    campus_id SERIAL PRIMARY KEY,
    campus_name VARCHAR(100) NOT NULL,
    address VARCHAR(100) NOT NULL
);

CREATE TABLE Program (
    program_id SERIAL PRIMARY KEY,
    program_name VARCHAR(250) NOT NULL,
    description TEXT NOT NULL
);

CREATE TABLE Class (
    class_id SERIAL PRIMARY KEY,
    class_name VARCHAR(10),
    start_date DATE,
    end_date DATE,
    program_id INTEGER REFERENCES Program(program_id),
    campus_id INTEGER NOT NULL REFERENCES Campus(campus_id)
);

CREATE TABLE Student (
    student_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    address VARCHAR(100) NOT NULL,
    email VARCHAR(250) NOT NULL
);

CREATE TABLE Student_Registration (
    registration_id SERIAL PRIMARY KEY,
    student_id INTEGER NOT NULL REFERENCES Student(student_id),
    class_id INTEGER NOT NULL REFERENCES Class(class_id),
    start_date DATE,
    end_date DATE,
    registration_status VARCHAR NOT NULL
);
```

Teknisk miljö

PostgreSQL + Docker

Jag har använt Docker Compose för att skapa en portabel, enkel och testbar miljö.

Allt finns konfigurerat i en docker-compose.yml och en .env-fil.

Två skript körs automatiskt vid uppstart:

– **01_create_tables.sql** (skapar schema och tabeller)

– **02_insert_data.sql** (laddar in fiktiv men realistisk data)

Detta ger i sin tur automatisering, versionskontroll och minimerar risken för fel.

Datainmatning

Fiktiv men realistisk data

Komplett databas innehållande:

- 3 program (Data Engineering, UX Design, Fullstack Development)
- 1 fristående kurs (API Development)
- 6 programklasser + 1 fristående klass
- 28 studenter, 7 instruktörer, 3 utbildningsledare
- 3 konsultföretag

Varje individ har:

- Ett **unikt ID**
- Ett personnummer
- Tydlig koppling till campus och klass

SQL Queries och Demo

Exempel på användning

Lista alla instruktörer och vilka kurser de undervisar:

```
yrkesco_db=# SELECT
    i.first_name,
    i.last_name,
    c.course_name,
    p.program_name,
    ic.teaching_period,
    ic.hours_allocated
FROM
    Instructor i
JOIN Instructor_Course ic ON i.instructor_id = ic.instructor_id
JOIN Course c ON ic.course_id = c.course_id
LEFT JOIN Program_Course pc ON c.course_id = pc.course_id
LEFT JOIN Program p ON pc.program_id = p.program_id
ORDER BY
    p.program_name;
 first_name | last_name |      course_name       |     program_name      | teaching_period | hours_allocated
------------+-----------+------------------------+-----------------------+-----------------+-----------------
 Oskar      | Ekman     | Database Modeling      | Data Engineering      | VT26            |              70
 Sara       | Nilsson   | Python for Data        | Data Engineering      | HT25            |              60
 Elin       | Sjöberg   | Web Development Basics | Fullstack Development | HT25            |              65
 Karin      | Holm      | Backend with Node.js   | Fullstack Development | VT26            |              75
 Ali        | Hassan    | Interaction Design     | UX Design             | VT26            |              55
 Jenny      | Bergström | UX Research Methods    | UX Design             | HT25            |              50
 Mats       | Thorsen   | API Development        |                       | HT25            |              60
(7 rows)
```

Affärsvärde

Vad tjänar YrkesCo på detta?

Denna databasmodell gör att YrkesCo kan **planera** sina program och kurser på ett effektivt sätt, underlätta i **administration** av studenter och personal samt **analysera** resursutnyttjande och kapacitet.

Modellen är flexibel för framtida tillväxt, förberedd för dashboards och fungerar som en stabil grund för ett UI eller ett API.

TACK!