Yrkeshögskola YrkesCo

YrkesCo är en framstående yrkeshögskola i Sverige med verksamhet på två campusorter:

- Göteborg
- Stockholm

Med framtida planer på expansion till fler orter är flexibilitet och skalbarhet nyckelfaktorer.

Vi tar sikte på att skapa en robust och säker datalösning som stödjer verksamhetens behov inom utbildning och personalhantering.

Nuvarande utmaningar:

Dålig dataintegritet

Otydlig datakonsistens leder till problem med tillförlitlighet och säkerhet.

Tidskrävande administration

Manuella processer bromsar effektiviteten och ökar risken för fel.

Svårigheter med översikt

Svårt att hålla koll på program, kurser och klasser för rapportering och planering.

Begränsad flexibilitet

Nuvarande system är inte skalbart för en växande organisation och expansion.

Business rules:

Datasekretess och separering
Allmän data och privata data lagras i separata tabeller.

Konsulter och företagskoppling
Konsulter är kopplade till ett företag via contracts.
Företag beskrivas med organisationsnummer,
f-skattsstatus, adress, telefonnummer, email, etc.

Begränsningar för utbildningsledare
Varje utbildningsledare ansvarar för maximalt tre klasser.

Campusadministration
YrkesCo har två anläggningar: Stockholm och Göteborg.
Systemet är skalbart för framtida expansion till fler orter.
Varje klass måste tillhöra en existerande campus.

Roller och anställdas struktur
Varje anställd tilldelas en roll. Konsulter jobbar som utbildare.

Program och Kursstruktur
Program beviljas i tre omgångar.
Program innehåller flera kurser.
Kurser kan också erbjudas som fristående kurser.

Kontrakt och undervisningsuppdrag
Kontrakt definierar konsultens timarvode, vilken campus de arbetar på samt start- och slutdatum.

Klass- och Studenthantering
Varje klass tillhör en campus.
Varje klass är kopplad till ett program eller fristående kurs.
En student kan antas endast till ett program men flera fristående kurser.

Identifierade entiteter

CAMPUS

PROGRAMS

COURSES

CLASSES

EMPLOYEES

TEACHING ASSIGNMENTS

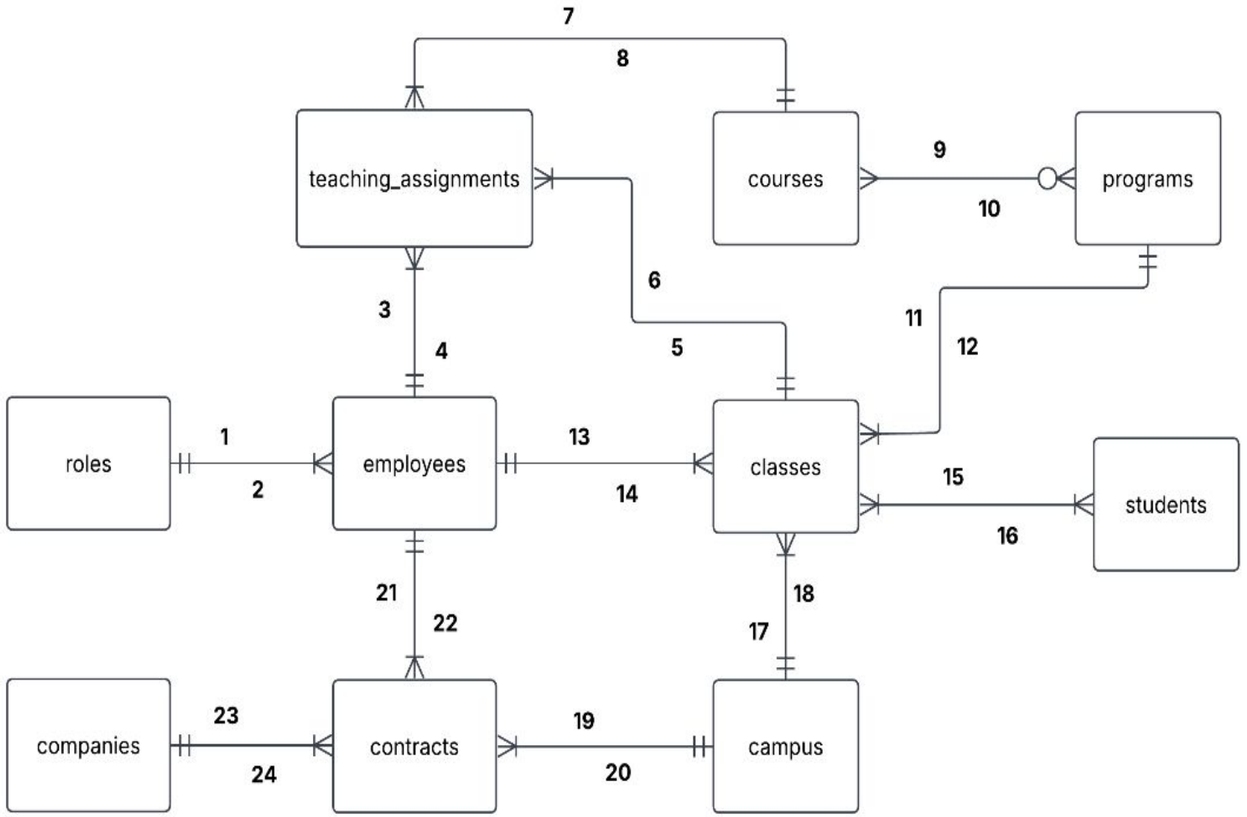
STUDENTS

ROLES

CONTRACTS

COMPANIES

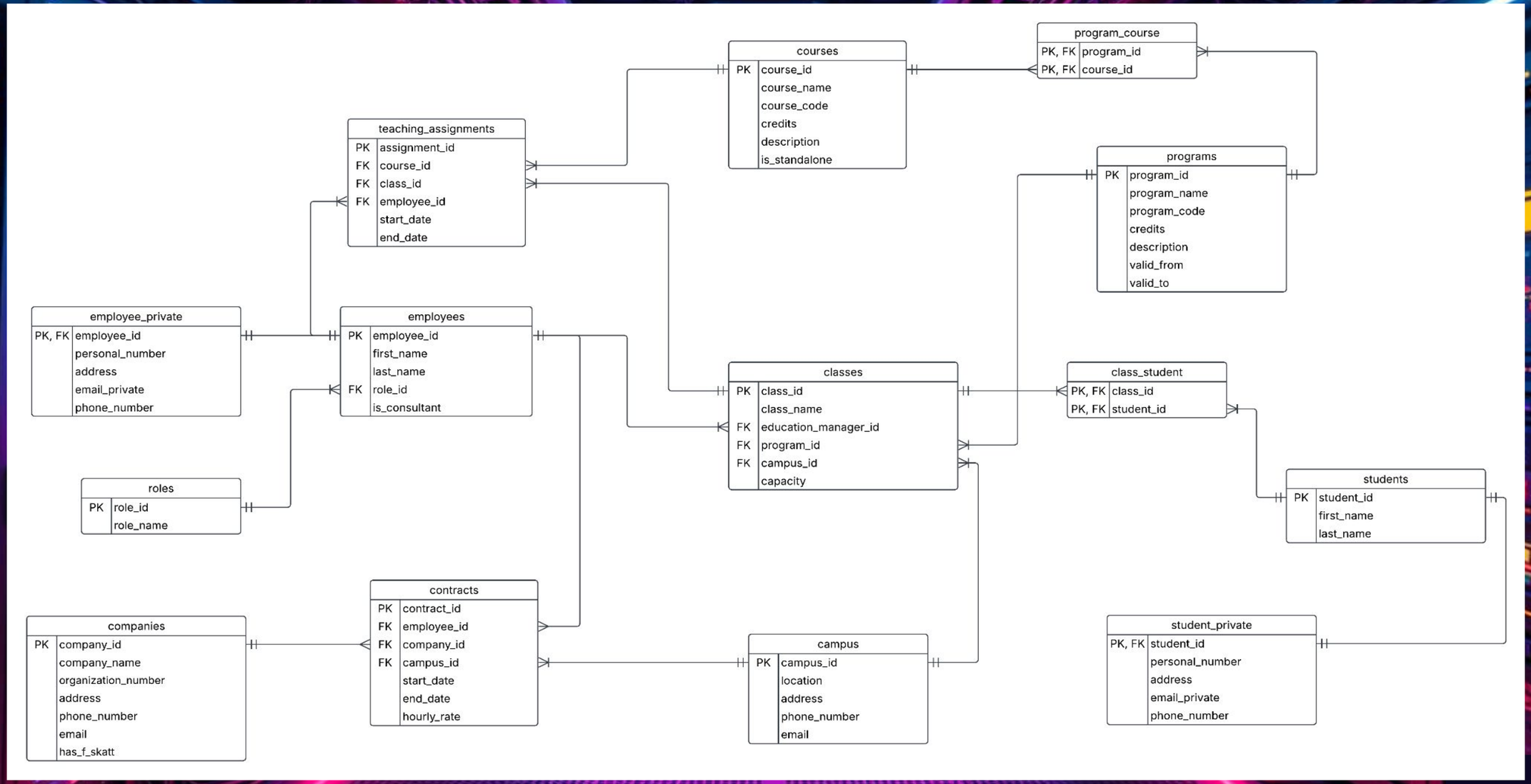
Konceptuell datamodell



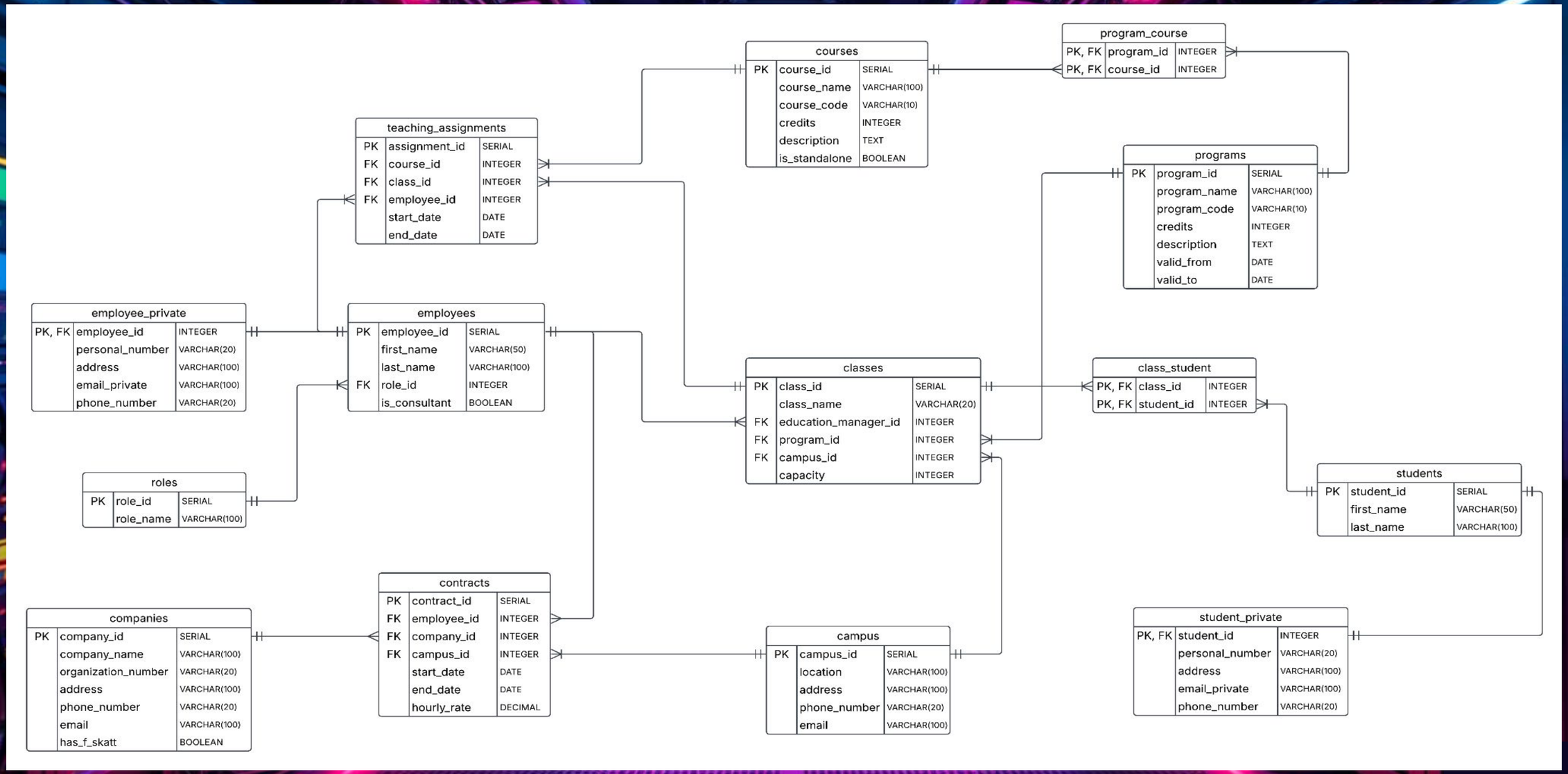
Description of relationships in the conceptual model

1. On one role hires one or many employees.
2. One employee can be hired for only one role.
3. One employee (teacher) can have one or many assignments.
4. Each teaching assignment belongs to only one teacher.
5. One class can have many teaching assignments.
6. Each teaching assignment can be associated with one class.
7. One course can have multiple entries in teaching assignments.
8. Each teaching assignment belongs to only one course.
9. One course can be a part of 0 (standalone) or many courses
10. One program has many courses
11. One program can contain many classes.
12. Each class is registered for only one program.
13. One educational manager (employee) can manage one or many classes (max 3 classes).
14. Each class has exactly one educational manager (employee).
15. One student can be registered in one or many classes
16. One class has one or many students
17. One campus has one or many classes
18. One class can be registered in only one campus
19. One campus can have many contracts.
20. Each contract applies to only one campus.
21. One employee (consultants) can be a party to several contracts.
22. Each contract belongs to only one employee (consultant).
23. One company can be a party to multiple contracts.
24. Each contract belongs to only one company.

Logisk datamodell

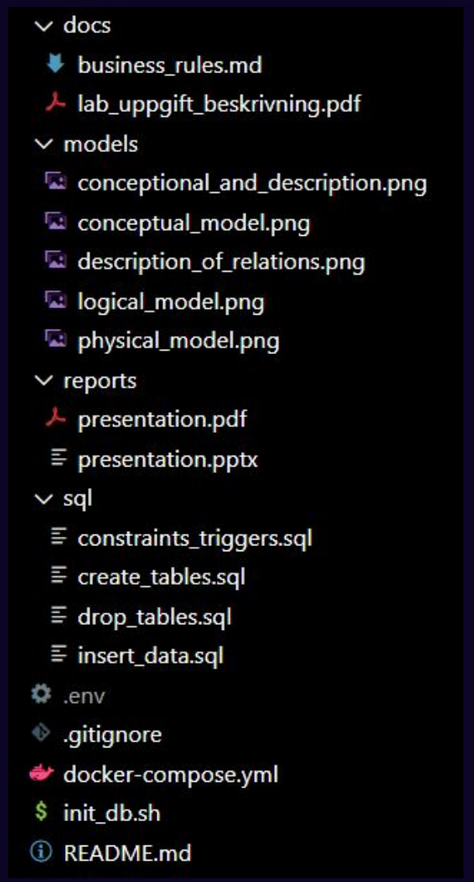


Fysisk datamodell



Fördjupning i struktur och kod

Projektets struktur: mappar, filer, bilder, models, etc.



Init_db.sh - startar projektet

$ init_db.sh

```
#!/bin/bash
set -e

CONTAINER_NAME="data_modeling_lab_yh"
DB_NAME="data_modeling_lab_yh"
DB_USER="postgres"

echo "🟡 Running a container..."
docker compose up -d

echo "⏳ Waiting for container to be ready..."
until docker exec "$CONTAINER_NAME" pg_isready -U "$DB_USER" -d "$DB_NAME" > /dev/null 2>&1; do
  sleep 2
done

echo "✅ The container is ready!"

echo "📂 Copying SQL files to container..."
docker cp "$(pwd)/sql" "$CONTAINER_NAME":/sql/

echo "🔥 Dropping all existing objects..."
docker exec -u postgres "$CONTAINER_NAME" bash -c "cd /sql && psql -d $DB_NAME -f drop_tables.sql"

echo "📦 Executing create_tables.sql..."
docker exec -u postgres "$CONTAINER_NAME" bash -c "cd /sql && psql -d $DB_NAME -f create_tables.sql"

echo "📦 Executing constraints_triggers.sql..."
docker exec -u postgres "$CONTAINER_NAME" bash -c "cd /sql && psql -d $DB_NAME -f constraints_triggers.sql"

echo "📦 Executing insert_data.sql..."
docker exec -u postgres "$CONTAINER_NAME" bash -c "cd /sql && psql -d $DB_NAME -f insert_data.sql"

echo "🎉 Done! Database initialized."
```

Det är dags att skapa tabeller

```sql
-- Roles table
CREATE TABLE IF NOT EXISTS "roles" (
  "role_id" SERIAL PRIMARY KEY,
  "role_name" VARCHAR(100) NOT NULL
);

-- Employees table (teachers, consultants, education managers, etc.)
CREATE TABLE IF NOT EXISTS "employees" (
  "employee_id" SERIAL PRIMARY KEY,
  "first_name" VARCHAR(50) NOT NULL,
  "last_name" VARCHAR(100) NOT NULL,
  "role_id" INTEGER NOT NULL,
  "is_consultant" BOOLEAN NOT NULL DEFAULT FALSE,
  CONSTRAINT "FK_employees.role_id"
    FOREIGN KEY ("role_id") REFERENCES "roles"("role_id")
);

-- Employees (Private information)
CREATE TABLE IF NOT EXISTS "employee_private" (
  "employee_id" INTEGER PRIMARY KEY,
  "personal_number" VARCHAR(20) NOT NULL UNIQUE,
  "address" VARCHAR(100) NOT NULL,
  "email_private" VARCHAR(100) NOT NULL,
  "phone_number" VARCHAR(20) NOT NULL
);

-- Students (General)
CREATE TABLE IF NOT EXISTS "students" (
  "student_id" SERIAL PRIMARY KEY,
  "first_name" VARCHAR(50) NOT NULL,
  "last_name" VARCHAR(100) NOT NULL
);
```

```sql
-- Students (Private)
CREATE TABLE IF NOT EXISTS "student_private" (
  "student_id" INTEGER PRIMARY KEY,
  "personal_number" VARCHAR(20) NOT NULL UNIQUE,
  "address" VARCHAR(100) NOT NULL,
  "email_private" VARCHAR(100) NOT NULL,
  "phone_number" VARCHAR(20) NOT NULL
);

-- Educational programs
CREATE TABLE IF NOT EXISTS "programs" (
  "program_id" SERIAL PRIMARY KEY,
  "program_name" VARCHAR(100) NOT NULL,
  "program_code" VARCHAR(10) NOT NULL UNIQUE,
  "credits" INTEGER NOT NULL,
  "description" TEXT,
  "valid_from" DATE NOT NULL,
  "valid_to" DATE NOT NULL
);

-- Campuses
CREATE TABLE IF NOT EXISTS "campus" (
  "campus_id" SERIAL PRIMARY KEY,
  "location" VARCHAR(100) NOT NULL,
  "address" VARCHAR(100) NOT NULL,
  "phone_number" VARCHAR(20) NOT NULL,
  "email" VARCHAR(100) NOT NULL
);
```

```sql
-- Courses
CREATE TABLE IF NOT EXISTS "courses" (
  "course_id" SERIAL PRIMARY KEY,
  "course_name" VARCHAR(100) NOT NULL,
  "course_code" VARCHAR(10) NOT NULL UNIQUE,
  "credits" INTEGER NOT NULL,
  "description" TEXT,
  "is_standalone" BOOLEAN NOT NULL DEFAULT FALSE
);

-- Bridge: program_course
CREATE TABLE IF NOT EXISTS "program_course" (
  "program_id" INTEGER NOT NULL,
  "course_id" INTEGER NOT NULL,
  PRIMARY KEY ("program_id", "course_id"),
  CONSTRAINT "FK_program_course.program_id"
    FOREIGN KEY ("program_id") REFERENCES "programs"("program_id"),
  CONSTRAINT "FK_program_course.course_id"
    FOREIGN KEY ("course_id") REFERENCES "courses"("course_id")
);

-- Companies
CREATE TABLE IF NOT EXISTS "companies" (
  "company_id" SERIAL PRIMARY KEY,
  "company_name" VARCHAR(100) NOT NULL,
  "organization_number" VARCHAR(20) NOT NULL UNIQUE,
  "address" VARCHAR(100) NOT NULL,
  "phone_number" VARCHAR(20) NOT NULL,
  "email" VARCHAR(100) NOT NULL,
  "has_f_skatt" BOOLEAN NOT NULL
);
```

sql > constraints_triggers.sql

```sql
-- Limit: max 3 classes per education manager
CREATE OR REPLACE FUNCTION check_max_classes()
RETURNS TRIGGER AS $$
BEGIN
  IF (
    SELECT COUNT(*) FROM classes
    WHERE education_manager_id = NEW.education_manager_id
  ) >= 3 THEN
    RAISE EXCEPTION 'Education manager (ID=%) already manages 3 classes', NEW.education_manager_id;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_max_classes
BEFORE INSERT ON classes
FOR EACH ROW
EXECUTE FUNCTION check_max_classes();

-- Restriction: the program must be valid for exactly 3 years
CREATE OR REPLACE FUNCTION check_program_duration()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.valid_to - NEW.valid_from) BETWEEN 1095 - 30 AND 1095 + 30 THEN
    RETURN NEW;
  ELSE
    RAISE EXCEPTION 'Program must be valid for 3 years';
  END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_program_duration
BEFORE INSERT ON programs
FOR EACH ROW
EXECUTE FUNCTION check_program_duration();
```

Some queries

Lägger till ett ny anläggning (Uppsala, Sverige)

```
data_modeling_lab_yh=# SELECT
    campus_id, location, address
FROM
    campus;
 campus_id | location  |                address
-----------+-----------+---------------------------------------
         1 | Stockholm | Sveavägen 50, 111 34 Stockholm
         2 | Göteborg  | Kungsportsavenyen 21, 411 36 Göteborg
(2 rows)

data_modeling_lab_yh=# INSERT INTO campus (location, address, phone_number, email)
VALUES ('Uppsala', 'Drottninggatan 7, 753 10 Uppsala', '018123456', 'uppsala@yh.se');
INSERT 0 1
data_modeling_lab_yh=# SELECT
    campus_id, location, address
FROM
    campus;
 campus_id | location  |                address
-----------+-----------+---------------------------------------
         1 | Stockholm | Sveavägen 50, 111 34 Stockholm
         2 | Göteborg  | Kungsportsavenyen 21, 411 36 Göteborg
         3 | Uppsala   | Drottninggatan 7, 753 10 Uppsala
(3 rows)
```

Visar varje elev och klassen de är inskrivna i

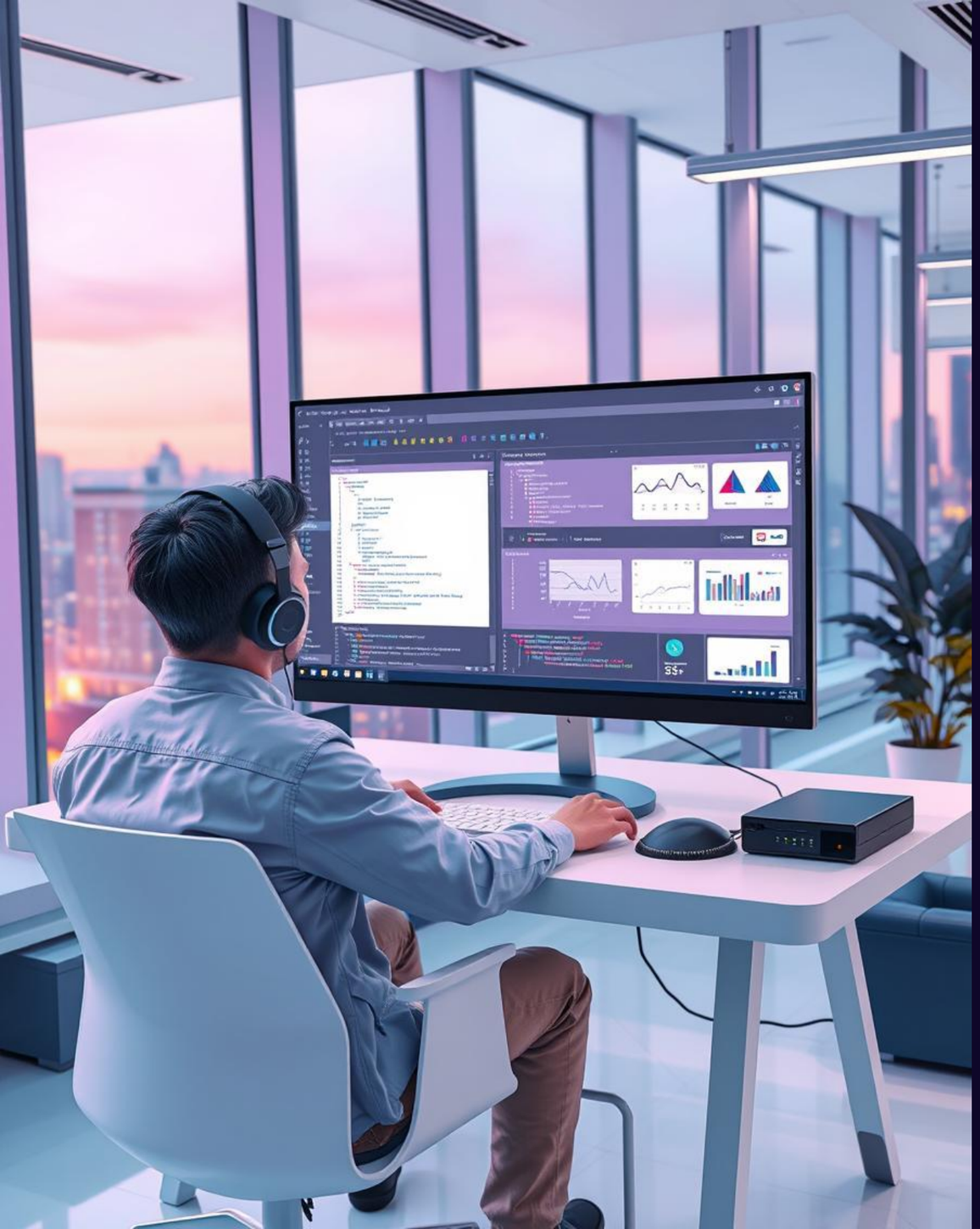
```
data_modeling_lab_yh=# SELECT
  s.first_name || ' ' || s.last_name AS "Student",
  cl.class_name AS "Class"
FROM
    students s
    JOIN class_student cs ON s.student_id = cs.student_id
    JOIN classes cl ON cs.class_id = cl.class_id
ORDER BY
    "Student"
LIMIT 10;
     Student      | Class
------------------+-------
 Adam Berglund    | DO401
 Agnes Nyström    | UX404
 Albin Borg       | MAD23
 Alexander Nyberg | SE102
 Alexander Öberg  | DO401
 Alice Lindgren   | DO401
 Alva Höglund     | DO401
 Amanda Söderlund | UX405
 Anton Hellgren   | UX405
 Daniel Forsberg  | UX404
(10 rows)
```

Visar hur många kurser varje lärare är tilldelad

```
data_modeling_lab_yh=# SELECT
  e.first_name || ' ' || e.last_name AS "Teacher",
  COUNT(DISTINCT ta.course_id) AS "Number of courses"
FROM teaching_assignments ta
JOIN employees e ON ta.employee_id = e.employee_id
GROUP BY e.employee_id, e.first_name, e.last_name
ORDER BY "Number of courses" DESC;
     Teacher      | Number of courses
------------------+-------------------
 Fredrik Olsson   |                 3
 Nils Borg        |                 2
 Karin Wikström   |                 2
 Quentin Sjöberg  |                 2
 Oskar Nyström    |                 1
 Anna Svensson    |                 1
 Rebecca Ström    |                 1
 Eva Johansson    |                 1
 Isabelle Persson |                 1
 Lars Åkesson     |                 1
(10 rows)
```

Visar varje program med antalet tilldelade kurser

```
data_modeling_lab_yh=# SELECT
    p.program_name AS "Program",
    COUNT(pc.course_id) AS "Number of courses"
FROM
    programs p
    LEFT JOIN program_course pc ON p.program_id = pc.program_id
GROUP BY
    p.program_name
ORDER BY
    "Number of courses" DESC;
        Program         | Number of courses
------------------------+-------------------
 Software Engineering   |                 5
 Data Science           |                 5
 Network Security       |                 4
 Cloud Computing        |                 3
 Mobile App Development |                 3
 UX/UI Design           |                 3
(6 rows)
```



1

Sammanfattning:

Designat och implementerat datamodell är säker, skalbar och anpassad till YrkesCos behov. Den hanterar studenter, utbildare, kurser, program, etc. – med tydliga roller och integritetsregler

2

Genomförda steg:

✔ Datamodellen har implementerats i en riktig PostgreSQL-databas

✔ Testdata har matats in (studenter, klasser, konsulter, avtal, m.m.)

✔ Funktionaliteten har testats via SQL-queries och triggers

3

Affärsnytta:

🚫 Färre Excelfiler

🔐 Bättre datasäkerhet

📈 Klar för framtida expansion

4

Tack för er tid!