YrkesCo DB

Database modeling of YrkesCo vocational school

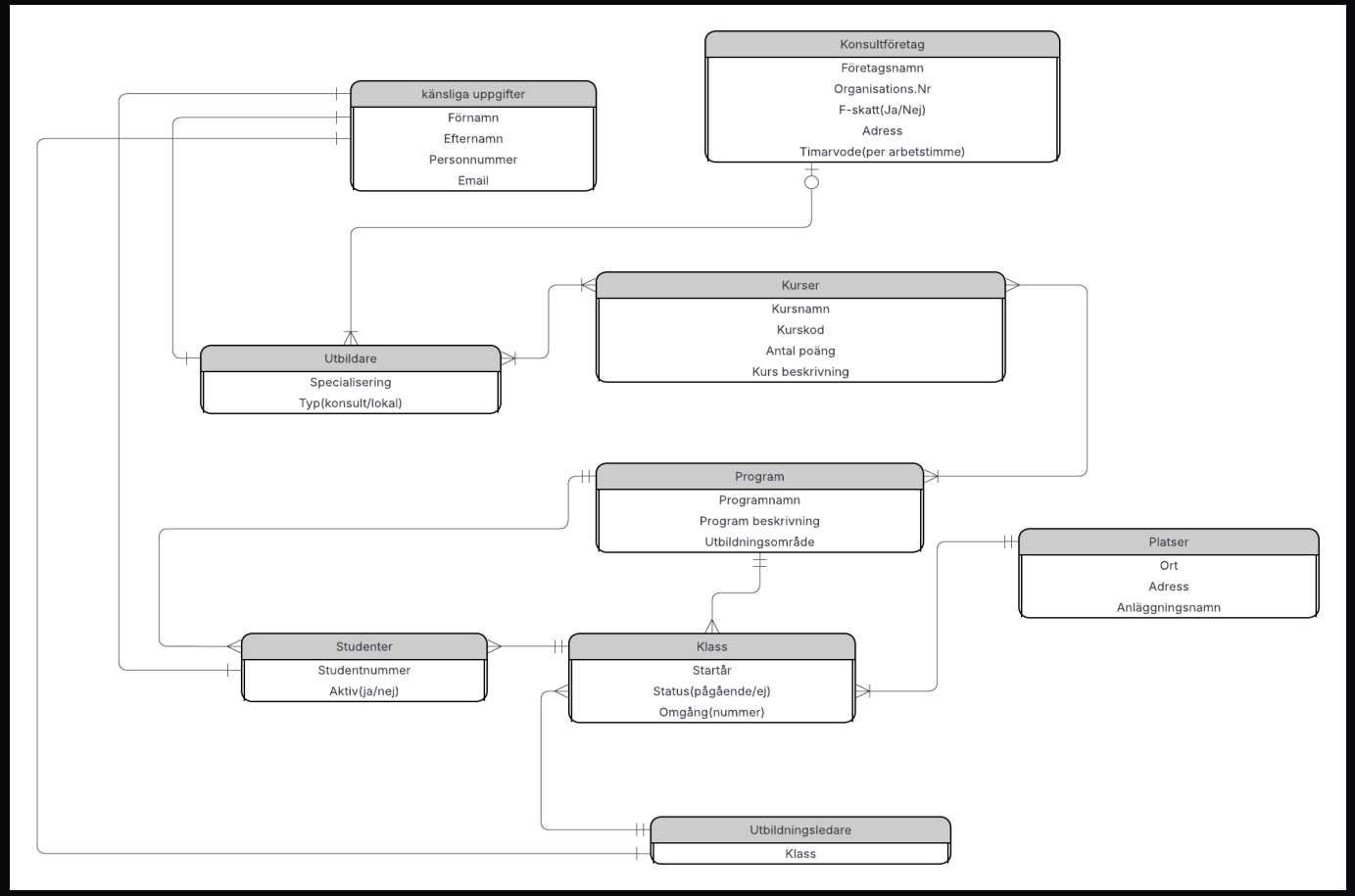
Introduktion:

YrkesCo Database:

Denna project och databas implementering av skolsystemet inom YrkesCo bestar av foljande 4 steg:

- Konceptuell modell
- Logisk modell
- Fysisk modell
- Implementation av de fysiska modellen

Konceptuell modell (Den ursprungliga idén av modellen)

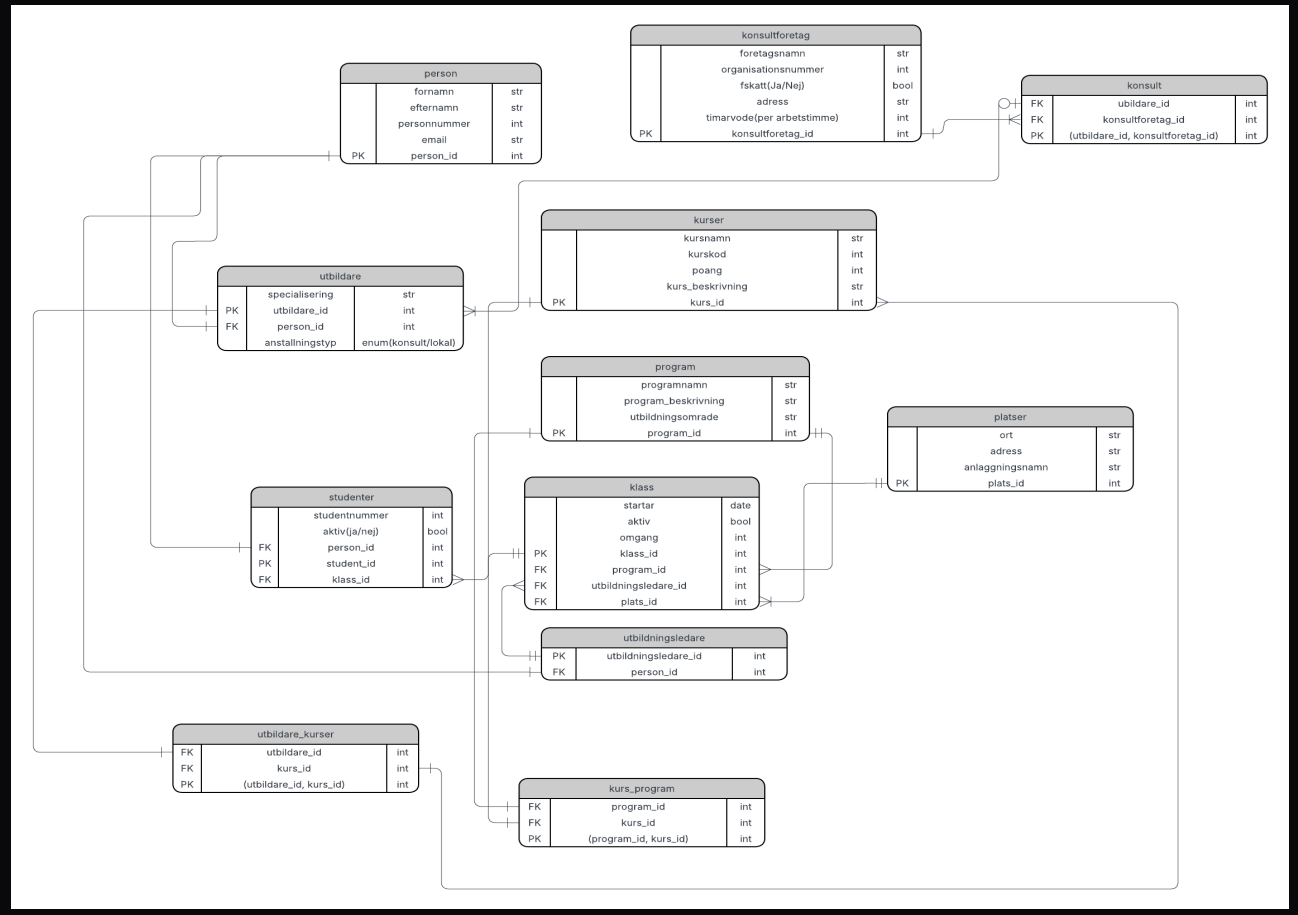


Den konceptuella modellens komponenter:

Den konceptuella modellen består av entiteter som håller information om skolsystemets olika delar:

- **Känsliga uppgifter:** Innehåller allt som rör uppgifter som möjligen kan identifiera individuella personer utanför deras roll inom skolsystemet.
- **Utbildare:** Både lärare och konsulter samlas här
- **Konsultföretag:** info om konsultföretagen som anlitas av skolan.
- **Program:** info om yrkeshögskolans program
- **Kurser:** info om skolans kurser
- **Klass:** info om de olika klasserna I yrkeshögskolan
- **Utbildningsledare:** info om utbildningsledarna för de olika klasserna.
- **Platser:** info om yrkeshögskolans anläggningar och vart de befinner sig.
- **Studenter:** studenternas alla uppgifter gällande skolan.

Logisk modell:



Logiska modellens komponenter(1):

Vår logiska modell introducerar nya attribut och 3 nya komponenter:

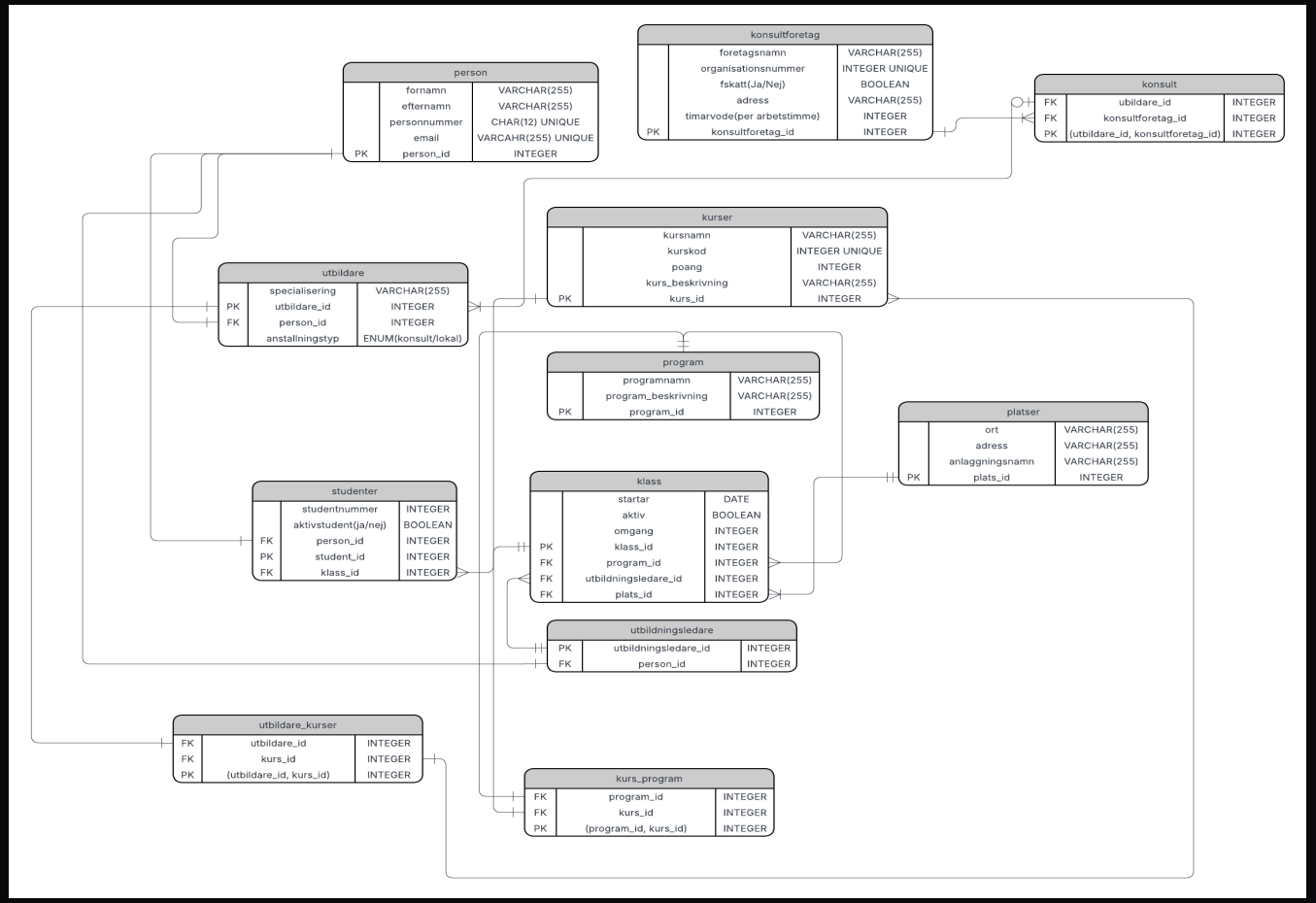
- **Konsult:** Tabell (junction table) för att kunna se alla utbildare som tillhör konsultföretag genom koppling av konsultföretag_id och utbildare_id.
- kurs_program: Tabell (junction table) för att koppla kurser til program samt kunna spåra kurser som är utan program.
- Utbildare_kurser: Tabell (junction table) för att koppla utbildare till kurser för att kunna se vilka utbildare som ansvarar för vilka kurser.

Logiska modellens komponenter(2):

Inom den logiska modellen är alla relationer mellan entiteterna etablerat med hjälp av motsvarande primär-nycklar (primary keys) och främmande-nycklar (foreign keys)

Samt markerar vi attribut med deras motsvarande data-typ:

Fysisk modell:



Den fysiska modellens komponenter:

Fysiska modellen:

- Beskriver de faktiska namn på de olika data-typer gällande korrekt språk format och deras karaktärs mängd
- Beskriver vilka attribut som behöver stämplas som "UNIQUE" för att förhindra dataredundans (data redundancy)
- Introducerar begränsningar (constraints) för korrekt inmatning av uppgifter och data.

Implementation av den fysiska modellen(Skapande):

För att implementerea den fysiska modellen börjar vi med att skriva tabellerna I en sql fil:

1. Vi skapar databasens struktur (schema) genom att skapa alla följande entiteter som tabeller I en PostgreSQL och Docker instans eller en sql fil.
2. Exempelvis blir entiteten "program" skapad på följande sätt:

```
CREATE TABLE program (
  programnamn VARCHAR(255) NOT NULL,
  programbeskrivning VARCHAR(255) NOT NULL,
  program_id SERIAL PRIMARY KEY
    );
```

3. Efter att vi har skapat våran databas struktur kopplar vi den med den faktiska datan.

Implementation av den fysiska modellen(Inmatning):

Vi matar in alla nödvändiga filer för att mata in data och bläddra genom den:

Vi kopplar följande filer till vår doker container med hjälp av PostgreSQL:

1. Schema.sql --> Databasens grundläggande struktur
2. Insert.sql --> Inmatning av verklig data (Även källa för realtidsdata)
3. Queries.sql --> Våra skrivna queries (bläddringar) som spottar den data vi önskar att se.

Query Demonstration(1):

Vi matar in falsk data och använder sedan en enkel query-fil för att testa processen:

Följande snitt från vår falsk-data:

```
INSERT INTO skol_info.person (fornamn, efternamn, personnummer, email)
VALUES
('Johan', 'Jorin', '848475746362', 'johan@gmail.com'),
('Anna', 'Svensson', '876543657689', 'ansve@gmail.com'),
('Janne', 'Hakansson', '787463526354', 'janne@gmail.com'),
('Mange', 'Jansoon', '989876453647', 'mange@gmail.com'),
('Peter', 'person', '897865746574', 'peter@gmail.com');

INSERT INTO skol_info.program (programnamn, program_beskrivning)
VALUES
('Natur', 'Natur programmet handlar om naturvetenskap hsffsfsbfbfbfbfbfjbbfjdbfbf sbfd bb jbjdb bf'),
('Ekonomi', 'lorem ipsun dsdnsdjdnnjsbjhbfsfbjfbf'),
('Teknik', 'sfhskffnkfjnfkfnkfdfndkjfknsfkdnfk');

INSERT INTO skol_info.platser (ort, adress, anlaggningsnamn)
VALUES
('stockhlm', 'lorem ipsum gatan 1', 'yrkesco 1'),
('goteborg', 'lorem ipsun gatan 2', 'yrkesco 2');

INSERT INTO skol_info.kurser (kusrnamn, kurskod, poang, kurs_beskrivning)
VALUES
('Matte', 858, 40, 'hbffbsffbjfbfbfbfjsbfjsbfdsbfjsbfs'),
('python', 834, 40, 'hdsfffjfbbsdbfjdssdbjfbjfbjfjfbj');
```

Query Demonstration(2):

Följande query fil:

```
--Vilken utbildningsledare har den har klassen?
SELECT
skol_info.person.fornamn,
skol_info.person.efternamn,
skol_info.klass.klass_id
FROM
skol_info.klass

JOIN
skol_info.utbildningsledare
ON skol_info.klass.utbildningsledare_id = skol_info.utbildningsledare.utbildningsledare_id

JOIN
skol_info.person
ON skol_info.person.person_id = skol_info.utbildningsledare.person_id

WHERE skol_info.klass.klass_id = 1;
```

Query Demonstration(3):

Vi matar in query filen I vår PostgreSQL och Docker instans:

```
postgres=# \i /queries.sql
```

Resultat från vår query:

```
 fornamn | efternamn | klass_id
---------+-----------+----------
 Johan   | Jorin     |        1
(1 row)
```