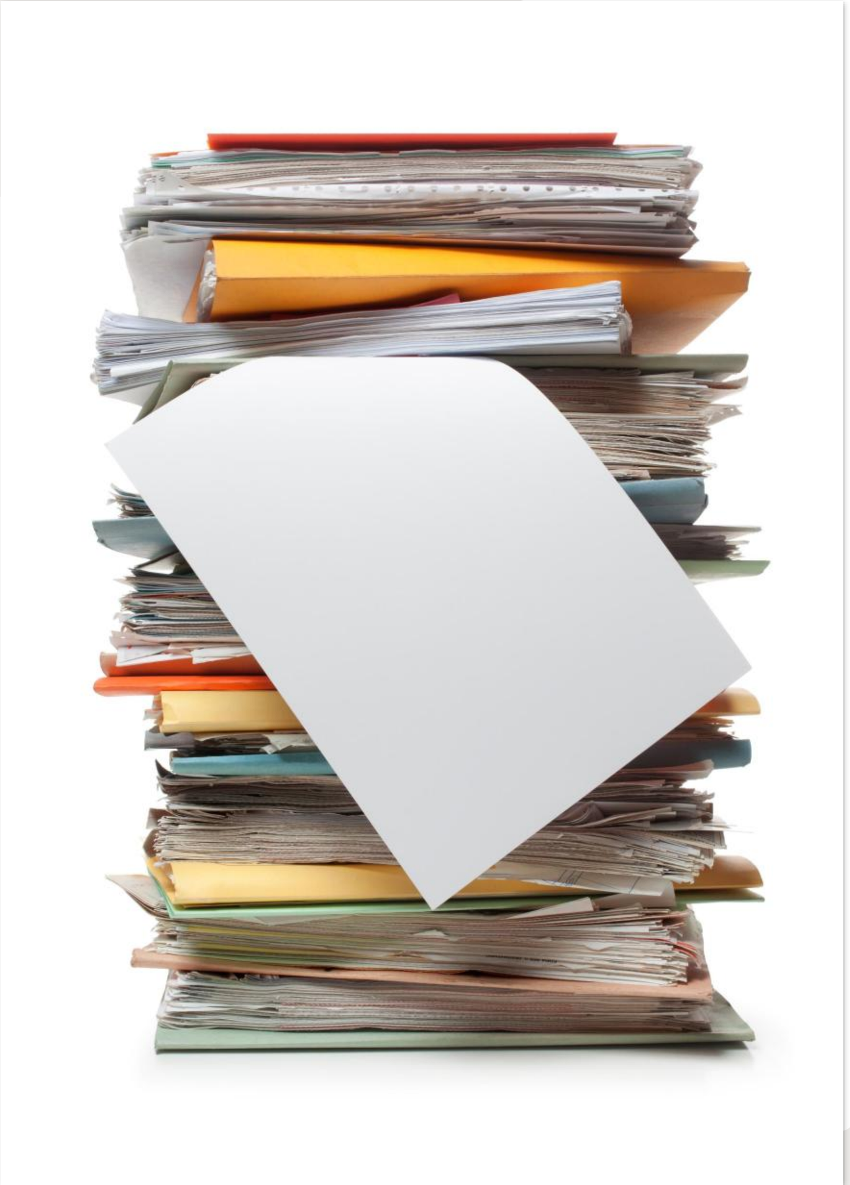
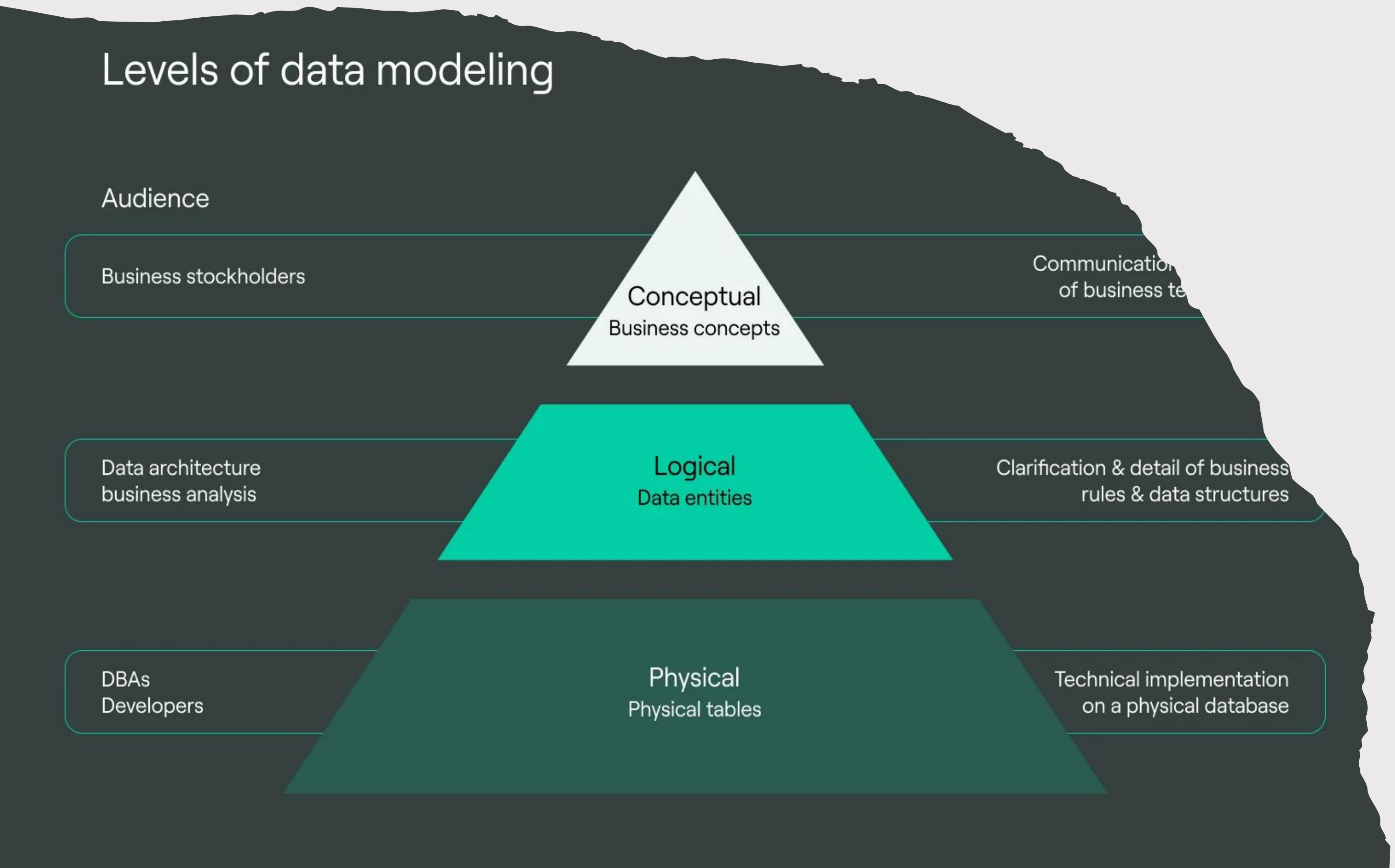
Yrkeshögskolan - YrkesCo

A database for YrkesCo, a vocational college, to replace scattered Excel files and streamline data management for students, educators, program leaders, courses, and consultants.



Conceptual to Physical model



Levels of data modeling

Audience

Business stockholders

Conceptual
Business concepts

Communicatio
of business te

Data architecture
business analysis

Logical
Data entities

Clarification & detail of business
rules & data structures

DBAs
Developers

Physical
Physical tables

Technical implementation
on a physical database

- **Reduces Costs and Risks**
  By identifying gaps and conflicts early in the planning phase, we can avoid costly changes later on.

- **Prevents Data Redundancy**
  Ensures that information isn't unnecessarily stored in multiple places, reducing the risk of inconsistencies.

- **Ensures Data Integrity & Consistency**
  Clearly defined relationships and rules help guarantee that data remains accurate and up to date.

- **Facilitates Scalability & Troubleshooting**
  A well-thought-out model makes it easier to scale, adapt, or debug the database in the future.

Conceptual model

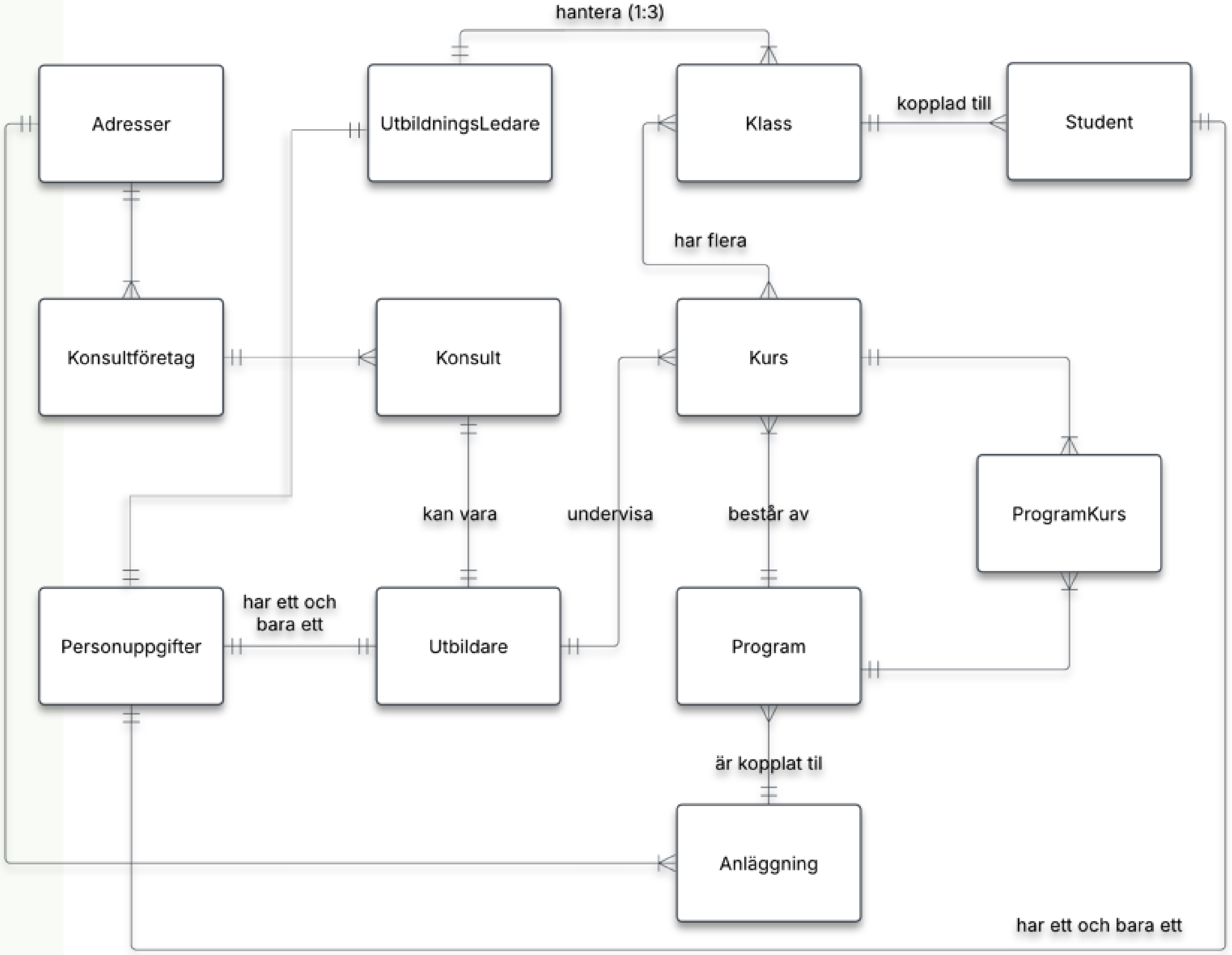
- Create a Separate Entity for Personal Information

- Introduce a Dedicated Address Table

- Add a Table for Consulting Companies

- Create Composite Entities for Many-to-Many Relationships

Entity-Relationship Diagram



Relationship statements:

PERSONUPPGIFTER kan tillhöra en och bara en **STUDENT, UTBILDARE eller UTBILDNINGSLEDARE**

Ett **KONSULTFÖRETAG** har exakt en **ADRESS**.

Men **ADRESSEN** kan användas av flera **KONSULTFÖRETAG**.

En **UTBILDARE** kan vara en **KONSULT**

En **KONSULT** är alltid en **UTBILDARE**.

En **KLASS** hanteras av exakt en **UTBILDNINGSLEDARE**.

En **UTBILDNINGSLEDARE** kan ansvara för upp till tre **KLASSER**.

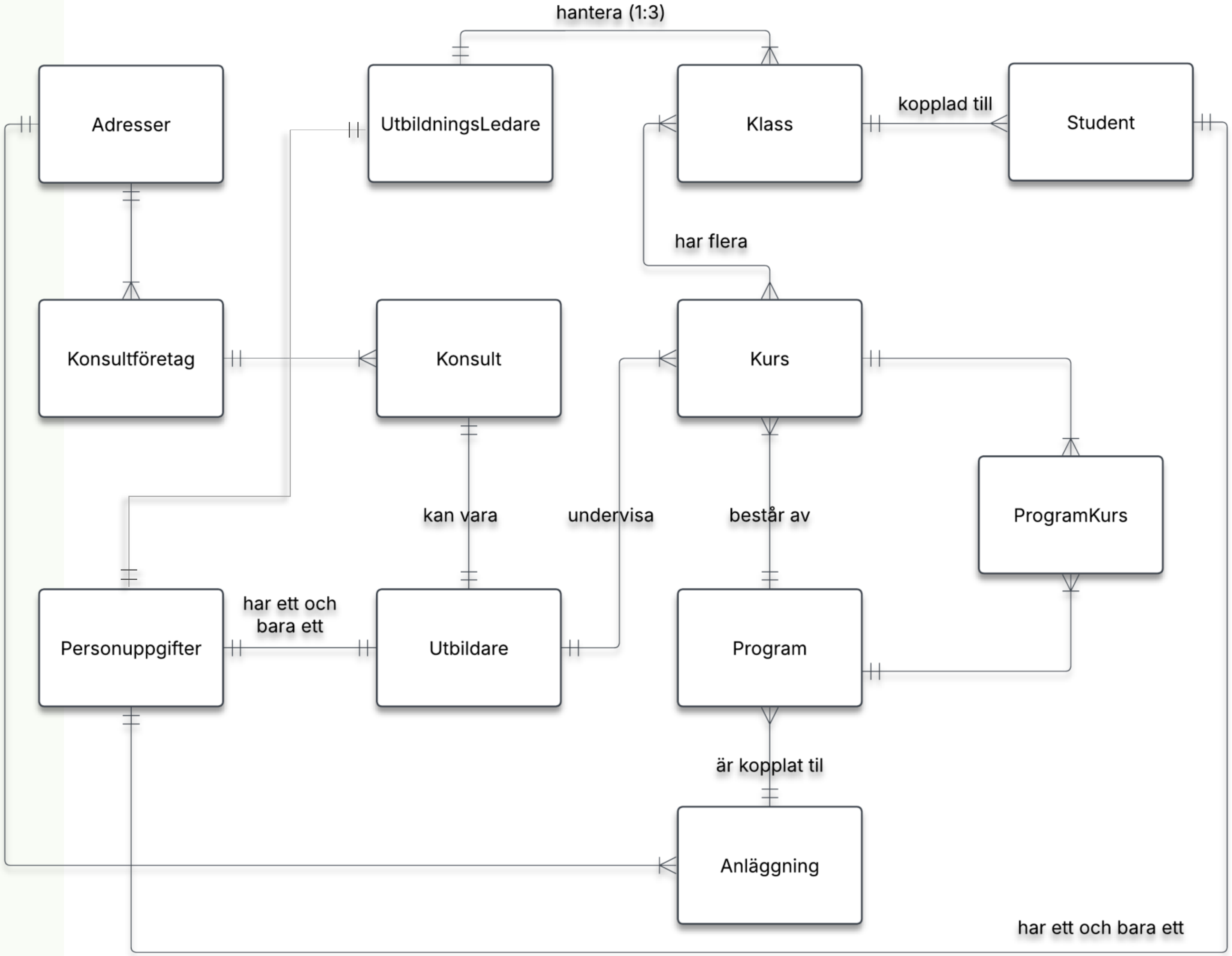
En **KLASS** tillhör exakt ett **PROGRAM**.

Ett **PROGRAM** består av tre **KLASSER**.

Ett **PROGRAM** består flera **KURSER** genom **PROGRAMKURS**.

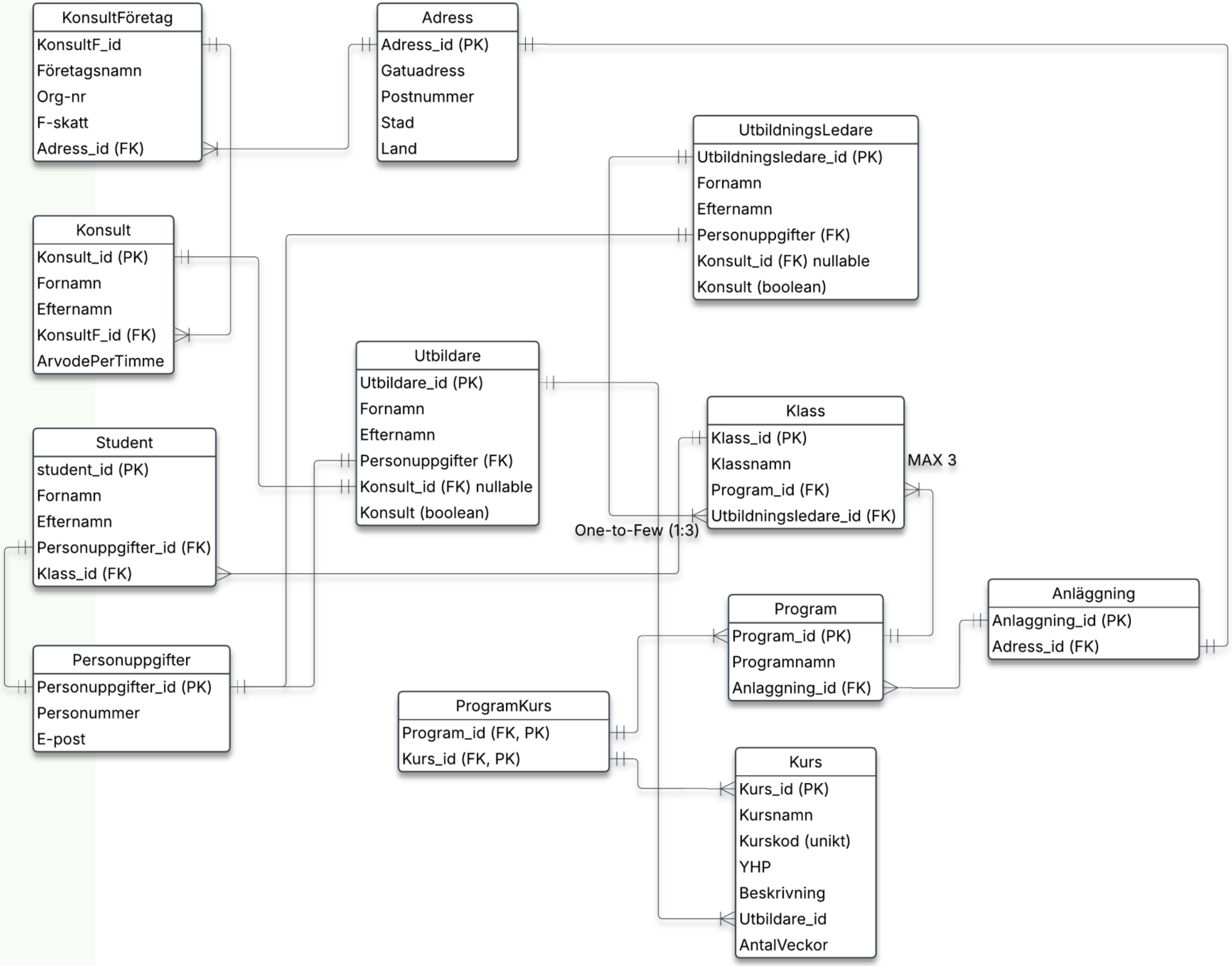
En **KURS** kan vara del av ett eller flera **PROGRAM** genom **PROGRAMKURS**.

Entity-Relationship Diagram



Logical model

- **Logical Model**: Intermediate phase between conceptual and physical models.
- **Entities & Attributes:** Convert business needs into detailed structures with attributes (columns) representing entity properties (e.g., student name, course code).
- **Primary & Foreign Keys:** Assign primary keys for unique identification and foreign keys to create relationships.
- **Cardinality:** Define the type of relationships (e.g., One-to-Many, One-to-One).
- **Data Types:** Begin identifying appropriate data types for attributes.



KonsultFöretag
KonsultF_id
Företagsnamn
Org-nr
F-skatt
Adress_id (FK)

Adress
Adress_id (PK)
Gatuadress
Postnummer
Stad
Land

UtbildningsLedare
Utbildningsledare_id (PK)
Fornamn
Efternamn
Personuppgifter (FK)
Konsult_id (FK) nullable
Konsult (boolean)

Konsult
Konsult_id (PK)
Fornamn
Efternamn
KonsultF_id (FK)
ArvodePerTimme

Utbildare
Utbildare_id (PK)
Fornamn
Efternamn
Personuppgifter (FK)
Konsult_id (FK) nullable
Konsult (boolean)

Klass
Klass_id (PK)
Klassnamn
Program_id (FK)
Utbildningsledare_id (FK)

MAX 3

One-to-Few (1:3)

Student
student_id (PK)
Fornamn
Efternamn
Personuppgifter_id (FK)
Klass_id (FK)

Anläggning
Anlagging_id (PK)
Adress_id (FK)

Program
Program_id (PK)
Programnamn
Anlaggning_id (FK)

Personuppgifter
Personuppgifter_id (PK)
Personummer
E-post

ProgramKurs
Program_id (FK, PK)
Kurs_id (FK, PK)

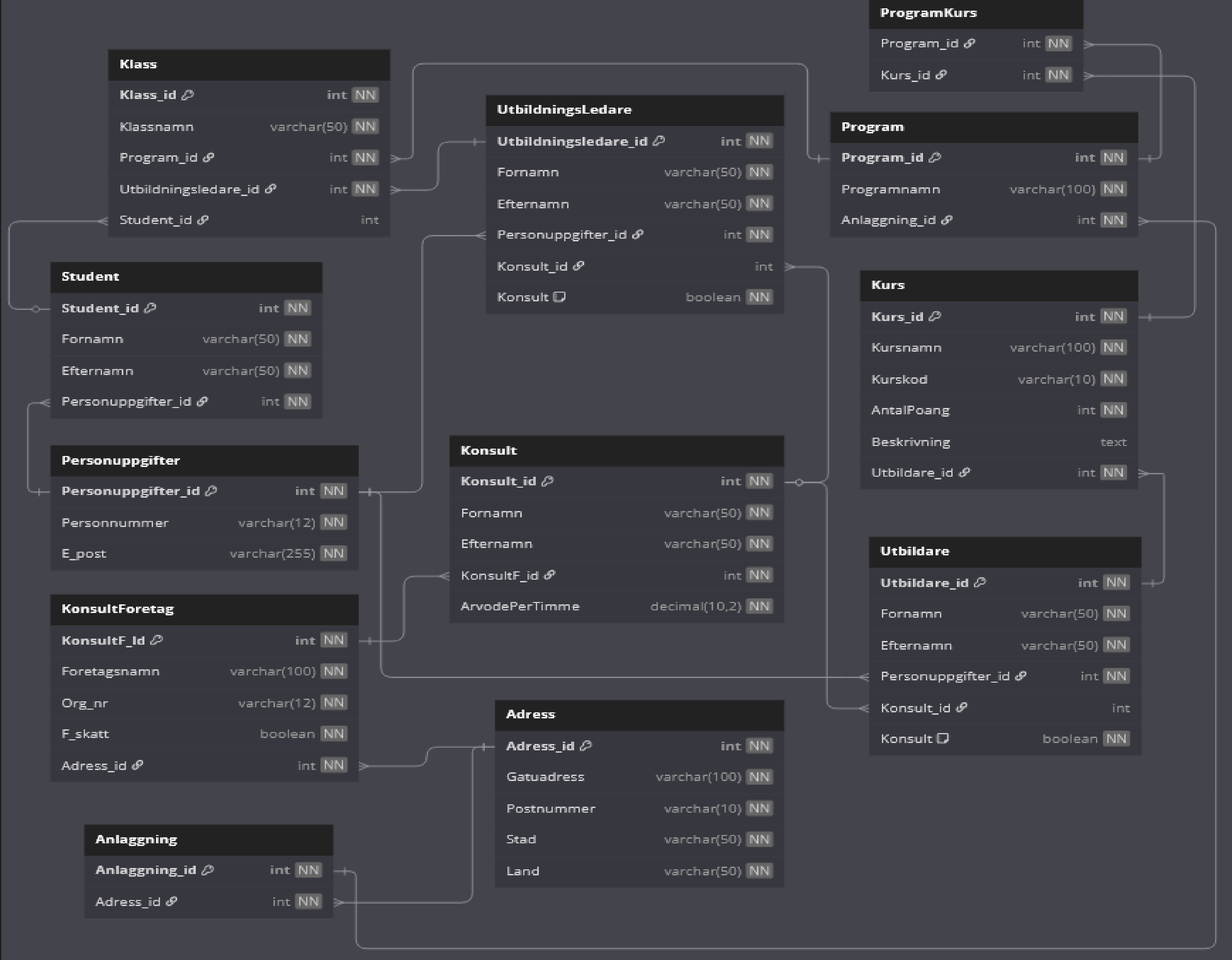
Kurs
Kurs_id (PK)
Kursnamn
Kurskod (unikt)
YHP
Beskrivning
Utbildare_id
AntalVeckor

Physical model

Physical Model: Final phase in data modeling.

DBML (Database Markup Language) : Used to define structure without direct database implementation.

Data Types & Constraints: Assign data types (e.g., VARCHAR, INTEGER) and define constraints



Klass		
Klass_id	int	NN
Klassnamn	varchar(50)	NN
Program_id	int	NN
Utbildningsledare_id	int	NN
Student_id	int	

Student		
Student_id	int	NN
Fornamn	varchar(50)	NN
Efternamn	varchar(50)	NN
Personuppgifter_id	int	NN

Personuppgifter		
Personuppgifter_id	int	NN
Personnummer	varchar(12)	NN
E_post	varchar(255)	NN

KonsultForetag		
KonsultF_Id	int	NN
Foretagsnamn	varchar(100)	NN
Org_nr	varchar(12)	NN
F_skatt	boolean	NN
Adress_id	int	NN

Anlaggning		
Anlaggning_id	int	NN
Adress_id	int	NN

UtbildningsLedare		
Utbildningsledare_id	int	NN
Fornamn	varchar(50)	NN
Efternamn	varchar(50)	NN
Personuppgifter_id	int	NN
Konsult_id	int	
Konsult	boolean	NN

Konsult		
Konsult_id	int	NN
Fornamn	varchar(50)	NN
Efternamn	varchar(50)	NN
KonsultF_id	int	NN
ArvodePerTimme	decimal(10,2)	NN

Adress		
Adress_id	int	NN
Gatuadress	varchar(100)	NN
Postnummer	varchar(10)	NN
Stad	varchar(50)	NN
Land	varchar(50)	NN

ProgramKurs		
Program_id	int	NN
Kurs_id	int	NN

Program		
Program_id	int	NN
Programnamn	varchar(100)	NN
Anlaggning_id	int	NN

Kurs		
Kurs_id	int	NN
Kursnamn	varchar(100)	NN
Kurskod	varchar(10)	NN
AntalPoang	int	NN
Beskrivning	text	
Utbildare_id	int	NN

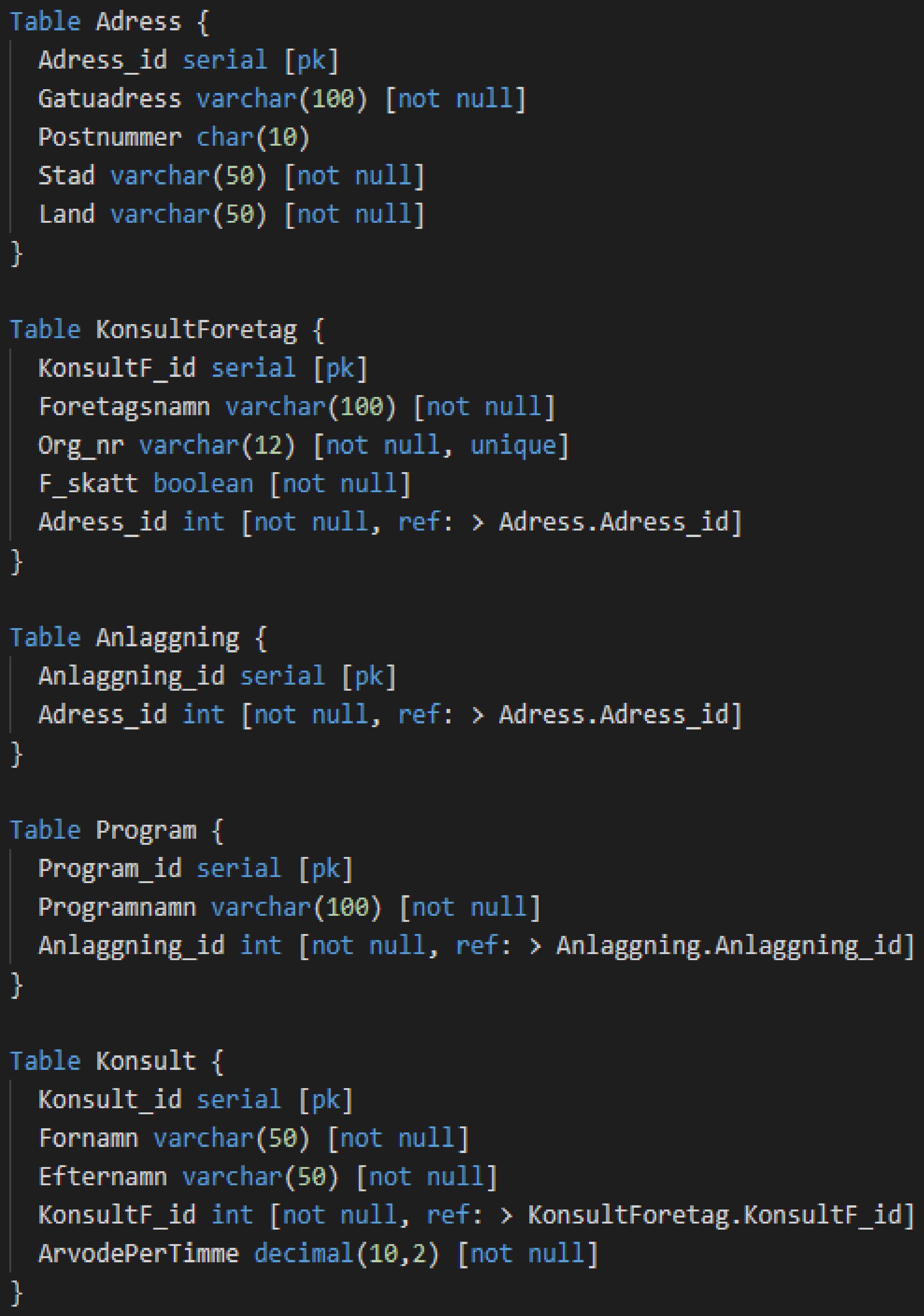
Utbildare		
Utbildare_id	int	NN
Fornamn	varchar(50)	NN
Efternamn	varchar(50)	NN
Personuppgifter_id	int	NN
Konsult_id	int	
Konsult	boolean	NN

Database Markup Language (DBML)

Data Types & Constraints: Attributes use data types like char, varchar, and serial. NOT NULL and UNIQUE ensure valid, unique values.

References & Relationships: ref creates relationships between tables, like KonsultForetag referencing Adress. References can be inline or at the end.

Composite Keys: Composite keys are defined with Indexes, combining primary keys from other tables, as seen in ProgramKurs.



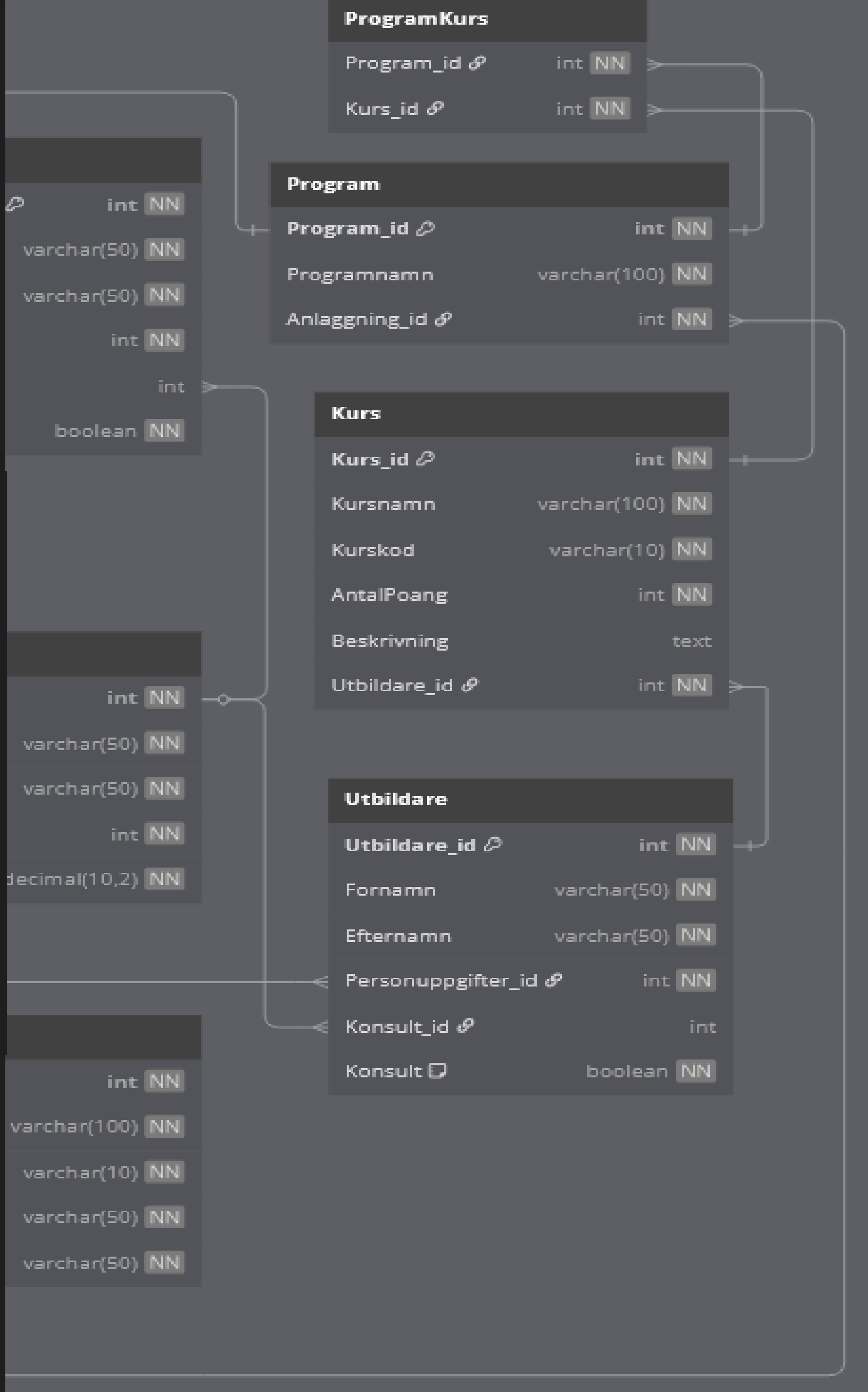
```
Table Adress {
  Adress_id serial [pk]
  Gatuadress varchar(100) [not null]
  Postnummer char(10)
  Stad varchar(50) [not null]
  Land varchar(50) [not null]
}

Table KonsultForetag {
  KonsultF_id serial [pk]
  Foretagsnamn varchar(100) [not null]
  Org_nr varchar(12) [not null, unique]
  F_skatt boolean [not null]
  Adress_id int [not null, ref: > Adress.Adress_id]
}

Table Anlaggning {
  Anlaggning_id serial [pk]
  Adress_id int [not null, ref: > Adress.Adress_id]
}

Table Program {
  Program_id serial [pk]
  Programnamn varchar(100) [not null]
  Anlaggning_id int [not null, ref: > Anlaggning.Anlaggning_id]
}

Table Konsult {
  Konsult_id serial [pk]
  Fornamn varchar(50) [not null]
  Efternamn varchar(50) [not null]
  KonsultF_id int [not null, ref: > KonsultForetag.KonsultF_id]
  ArvodePerTimme decimal(10,2) [not null]
}
```



ProgramKurs		
Program_id	int	NN
Kurs_id	int	NN

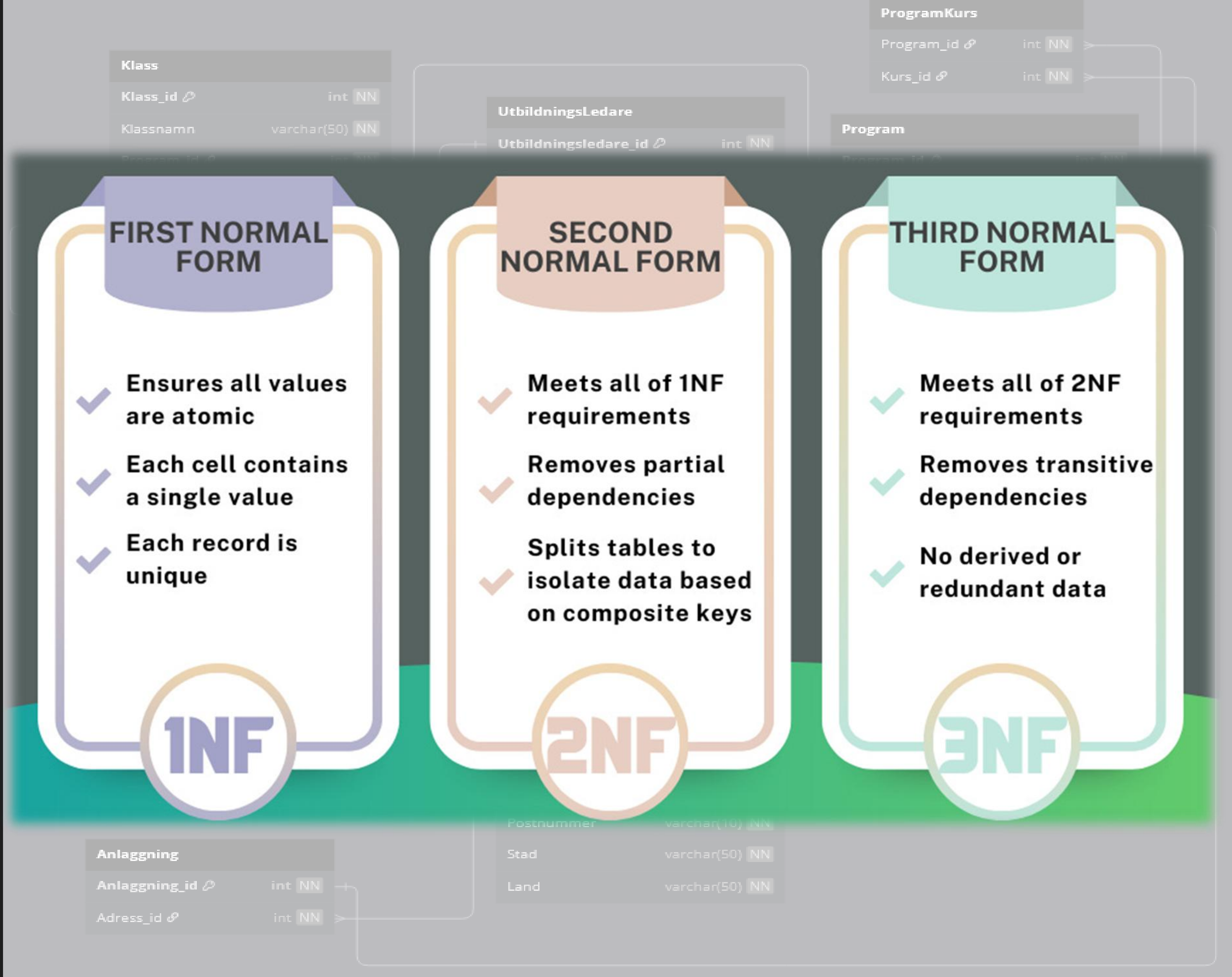
Program		
Program_id	int	NN
Programnamn	varchar(100)	NN
Anlaggning_id	int	NN

Kurs		
Kurs_id	int	NN
Kursnamn	varchar(100)	NN
Kurskod	varchar(10)	NN
AntalPoang	int	NN
Beskrivning	text	
Utbildare_id	int	NN

Utbildare		
Utbildare_id	int	NN
Fornamn	varchar(50)	NN
Efternamn	varchar(50)	NN
Personuppgifter_id	int	NN
Konsult_id	int	
Konsult	boolean	NN

Normalization – 3NF

- ✓ No transitive or partial dependencies
- ✓ All attributes depend on the primary key
- ✓ Handling many-to-many relationships with bridge tables
- ✓ Correct use of foreign keys and referential integrity



FIRST NORMAL FORM

- ✓ Ensures all values are atomic
- ✓ Each cell contains a single value
- ✓ Each record is unique

1NF

SECOND NORMAL FORM

- ✓ Meets all of 1NF requirements
- ✓ Removes partial dependencies
- ✓ Splits tables to isolate data based on composite keys

2NF

THIRD NORMAL FORM

- ✓ Meets all of 2NF requirements
- ✓ Removes transitive dependencies
- ✓ No derived or redundant data

3NF

```
irror_mod = modifier_ob.
t mirror object to mirror_
irror_mod.mirror_object
peration == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
_operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
_operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True

election at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier_
irror_ob.select = 0
bpy.context.selected_obj
ata.objects[one.name].se

int("please select exactly

- OPERATOR CLASSES ----

types.Operator):
X mirror to the selected
ject.mirror_mirror_x"
rror X"

context):
xt.active_object is not
```

Implement data PostgreSQL

QUERY!

Ex.

- ✓ Each program should have at least 3 courses attached. What program has what courses?
- ✓ Varje utbildningsledare ska vara ansvarig för 3 klasser. Vilken utbildningsledare är ansvarig för vilka klasser?
- ✓ Show student with the JOINED personuppgifts table
- ✓ What courses are connected to what programs?
- ✓ Ett program ska ha 3 klasser, vilka klasser är kopplade till vilket program?