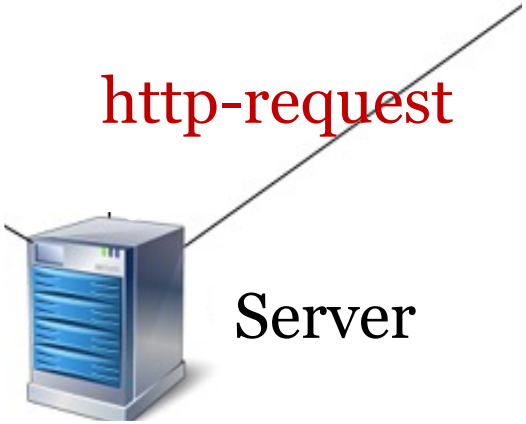
Databaser

TDDD80 Mobila och sociala applikationer

LINKÖPINGS UNIVERSITET

Labb S1 gick ut på server-kod och att skicka requests

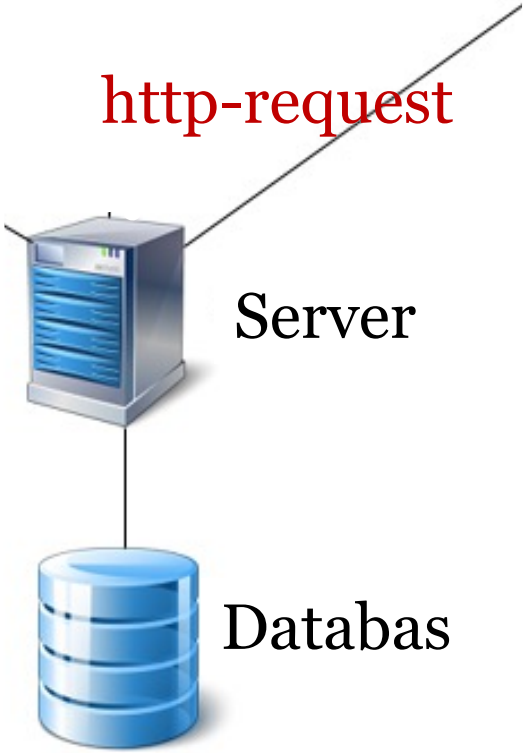
Fusk-klient



Labb S1: Server-kod

Nästa labb S2: använd databas

Fusk-klient



Labb S2: Databas

Översikt

- Relationsdatabaser (tabeller)
- SQL (Structured Query Language)
- Relationsmodeller
  - Olika typer av relationer
- SQLAlchemy
  - Stöd för att hantera SQL
- Enhetstestning

Olika typer av databaser

- Relationsdatabaser (SQL-databaser)
  - Fokus på entiteter (saker) och relationer mellan dessa
  - Tabeller
    - Varje rad måste vara unik
- NoSQL databaser
  - Dokument-orienterade databaser (t.ex. MongoDB)
  - JSON-databaser (t.ex. Firestore)
  - Key-value databaser (Redis)

Databas i labb2: SQLite

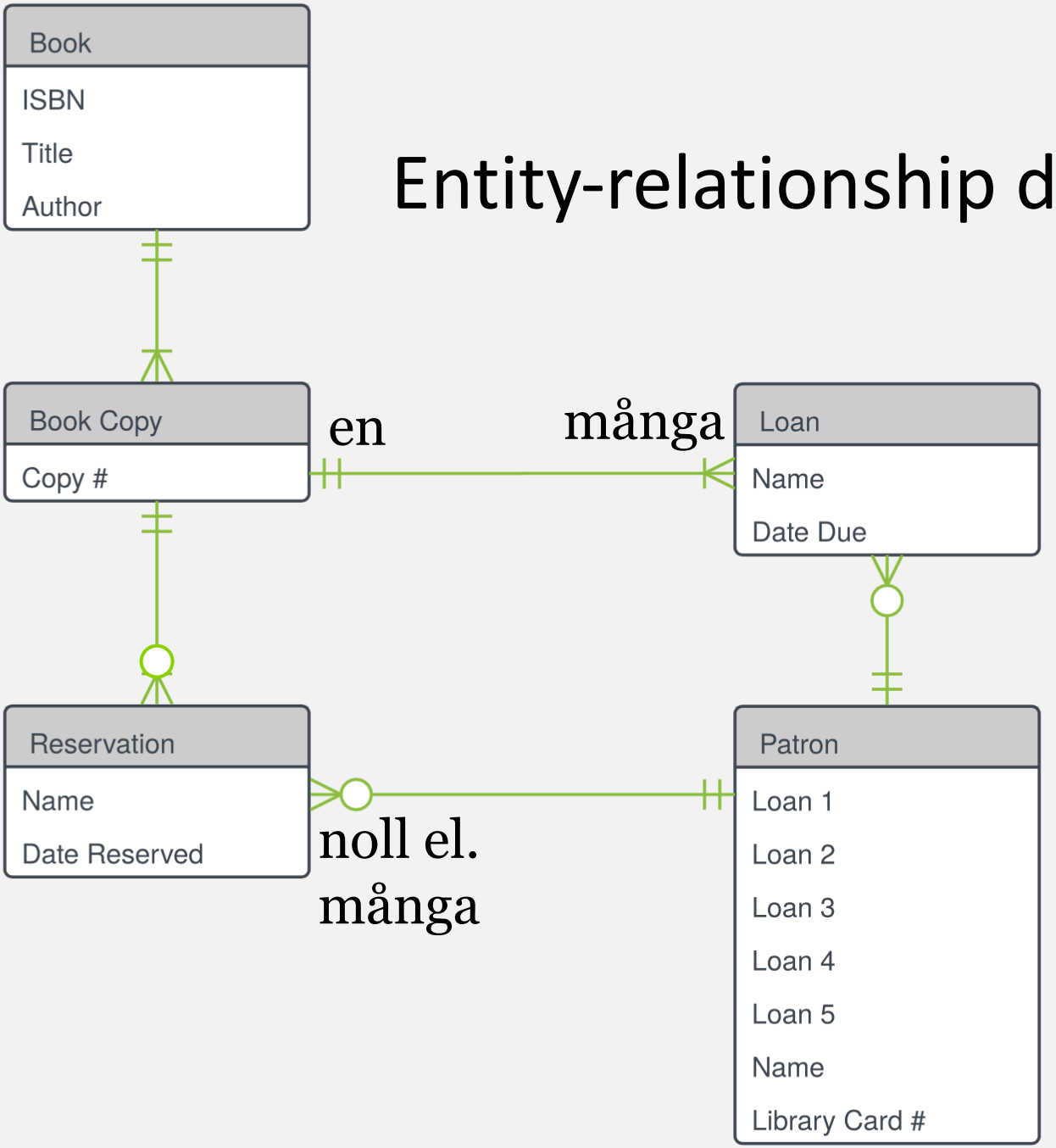
- Enkel relationsdatabas som kan lagras på en fil
  - Oftast lokalt på egen dator
  - En användare (klient)
  - Används med fördel för utveckling och testing
    - T.ex. lätt att rensa databasen genom att radera filen

Definition av tabeller

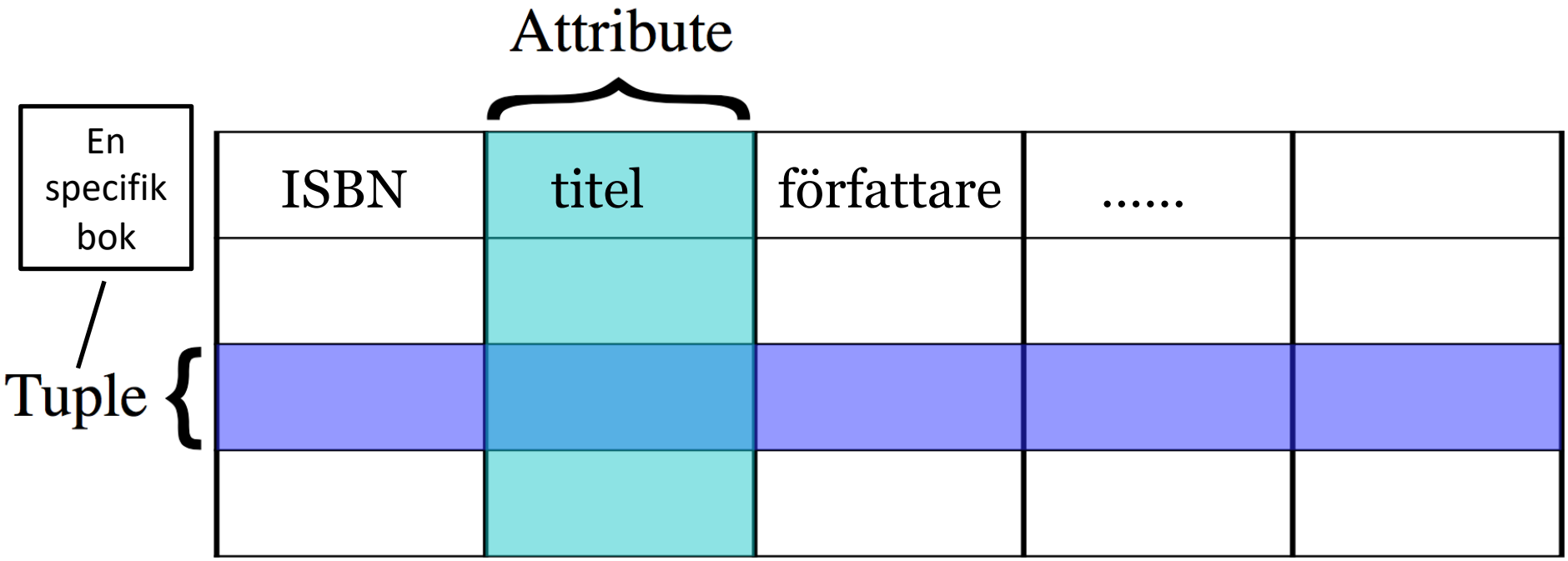
Ex: Biblioteksapp

- Vill lagra **låntagare**
  - Namn, lånekortsnummer
- Vilka **böcker** de har lånat
  - Låntagaren kan ha flera lån
- Vill lagra **böcker**
  - Författare
  - Utlånad datum
  - Reserverad datum

Entity-relationship diagram (ER)



Tabellen för ‘Böcker’

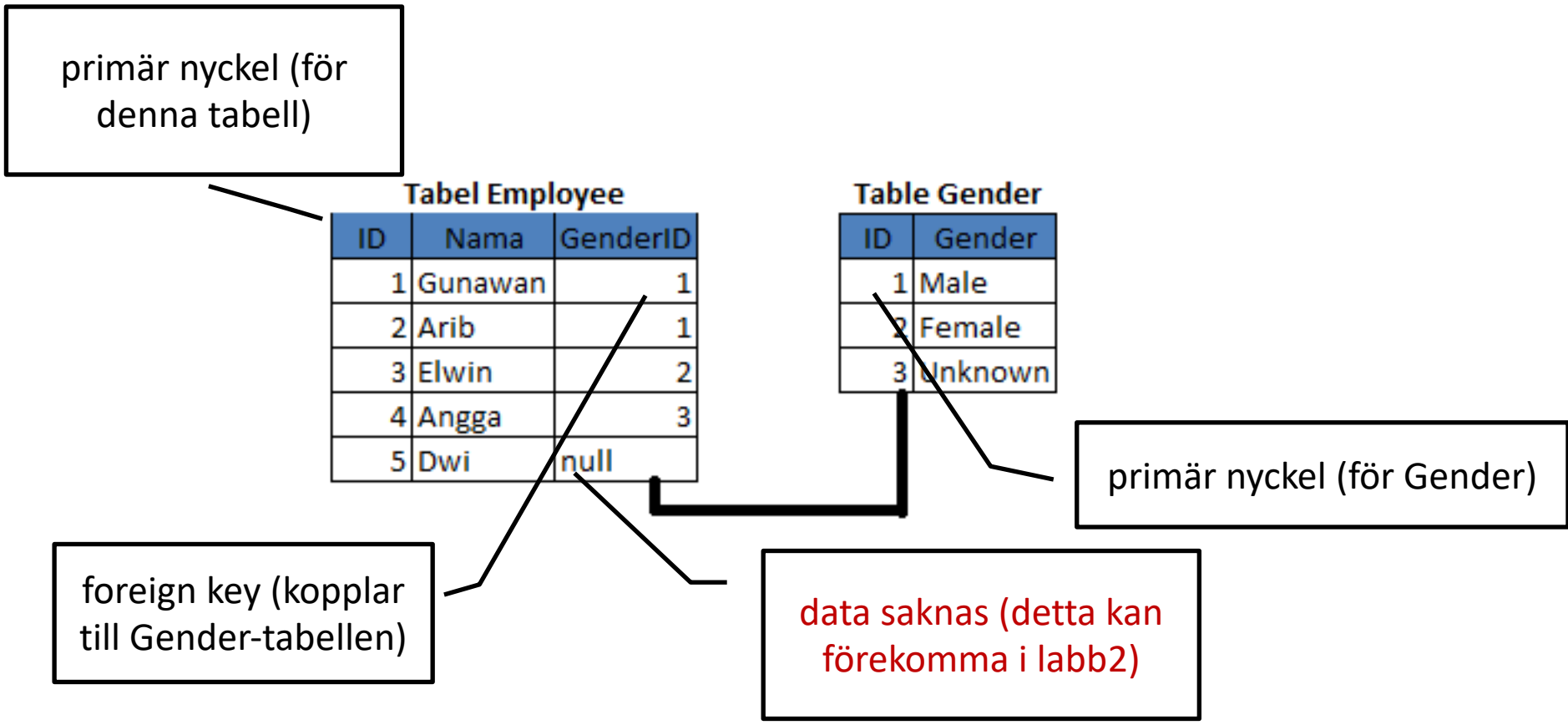


Varje rad måste vara unik!

Nycklar

- Varje tabell har en kolumn med primärnyckel (primary key, dvs. ID)
  - Används för att hitta raden
  - Unik för varje rad
- Främmande nyckel (foreign key)
  - Kolumn med referenser till annan tabells primärnyckel
  - T.ex. länk från bok (i Bok-tabellen) till författare (i Författare-tabellen)

Tabeller (exempel)



Hur vet servern vilka tabeller och kolumner som ska sättas upp?

Schema (“tabellhuvud”)

- Beskriver vilka kolumner som ingår i tabellen
- För varje kolumn
  - Kolumnnamn
  - Typ av tillåtna värden
    - Integer
    - String
  - Constraints
    - Unik
    - Non-null

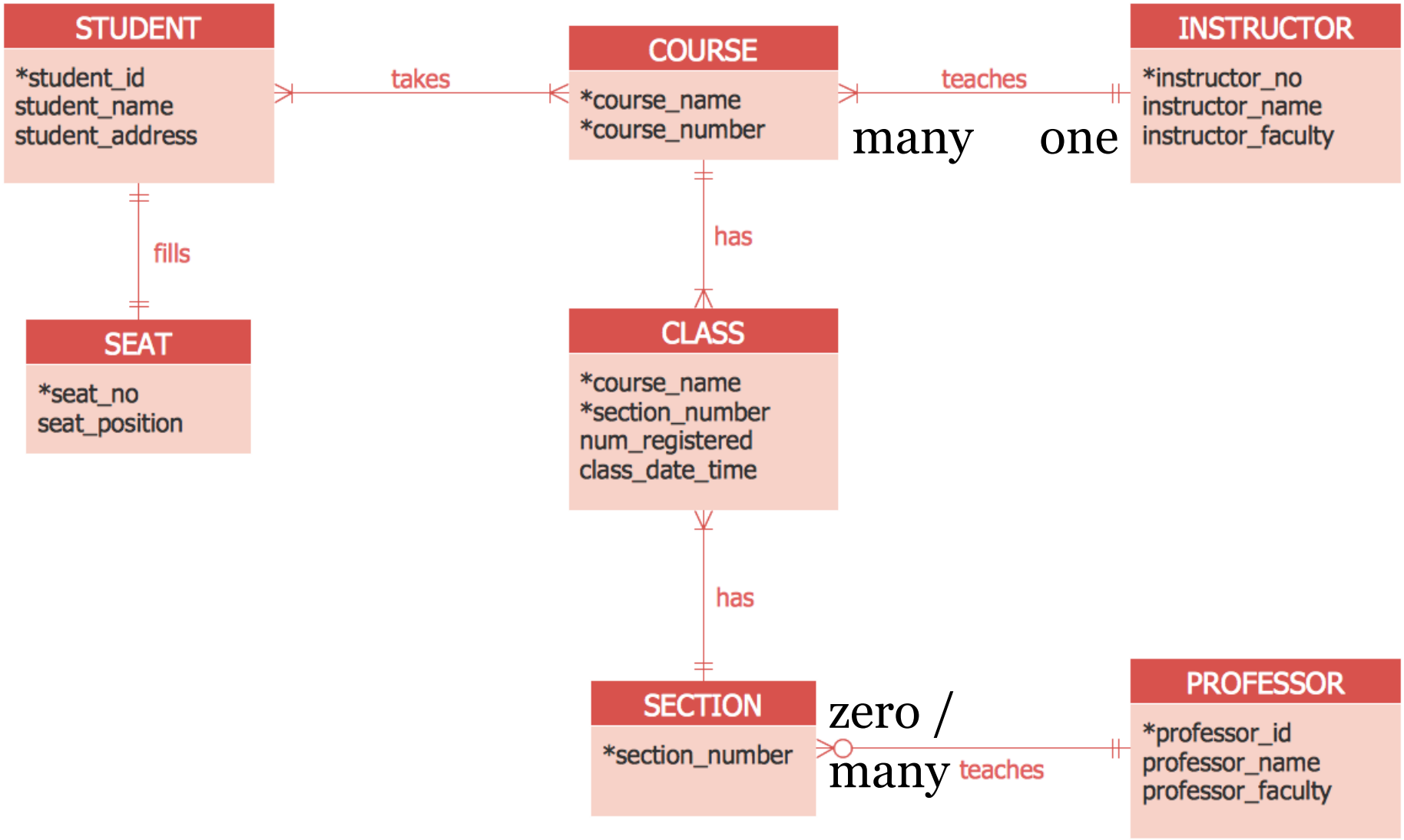
Tabellrader

Tabellrader (innehåll) skapas sedan dynamiskt när serverkoden körs och tabellen fylls på med data

Hur ska man bestämma schemat?

Relationsmodeller

ER (Entity-Relationship diagrams)


STUDENT
*student_id
student_name
student_address
takes
COURSE
*course_name
*course_number
teaches
many
one
INSTRUCTOR
*instructor_no
instructor_name
instructor_faculty
fills
has
SEAT
*seat_no
seat_position
CLASS
*course_name
*section_number
num_registered
class_date_time
has
SECTION
*section_number
zero /
many
teaches
PROFESSOR
*professor_id
professor_name
professor_faculty

Entiteter och relationer

- Entitet (“saker”)
  - Substantiv
- Relationer
  - Hur entiteter är kopplade
  - T.ex. Student (ent.) läser (rel.) Kurs (ent.)

Entiteter och relationer

- Relationer
  - One-to-one
    - Student – Adress
    - Student – Utbildningsplats (seat)
  - One-to-many
    - Kurs – Föreläsning
  - Many-to-one
    - Kurs – Kursansvarig
  - Many-to-many
    - Kurs – Studenter

Entiteter och relationer

- One-to-one
  - Kolumner i samma tabell
  - Två tabeller kopplade via foreign key
- Many-to-one, one-to-many
  - Två tabeller, där den ena länkar till den andra via foreign key
- Many-to-many
  - …

One-to-one

- Attribut för entitet: lägg i samma tabell

Låntagare #	Namn	Hemadress
1	Karl Karlsson	Rekrytgatan 1
2	Pär Svensson	Valhallavägen 34

- Olika entiteter: lägg i olika tabeller

Student ID	Namn	Utb. plats
1	Karl Karlsson	2
2	Pär Svensson	4

Utb.plats #	…
1	…
2	…
3	…
4	…

One-to-many

- Alla fält måste ha singulära värden, dvs. ej listor
  - Ex. samma låntagare kan låna flera böcker

Lån #	Namn	Lånat bok
1	Karl Karlsson	Krig och fred
2	Karl Karlsson	Blomskötsel

- Men, bättre med två tabeller:
  - En tabell för Bok
  - En tabell för Låntagare

One-to-many, many-to-one

- Varje entitet i egen tabell
- Fördelar:
  - Tydligare, mer lättläst
  - Mindre minneskrav

Lite bättre, men…

- Fortfarande dubblettrader

Lån #	Namn	Lånat bok
1	Karl Karlsson	2
2	Karl Karlsson	4

Bok #	Titel	Författare
1	Krig och fred	Tolstoy, Leo
2	…	
3	…	
4	Blomskötsel	Jönsson, Pär

- Måste söka igenom hela tabellen för att ta bort låntagare som har flyttat från stan

Gå från *many* till *one*

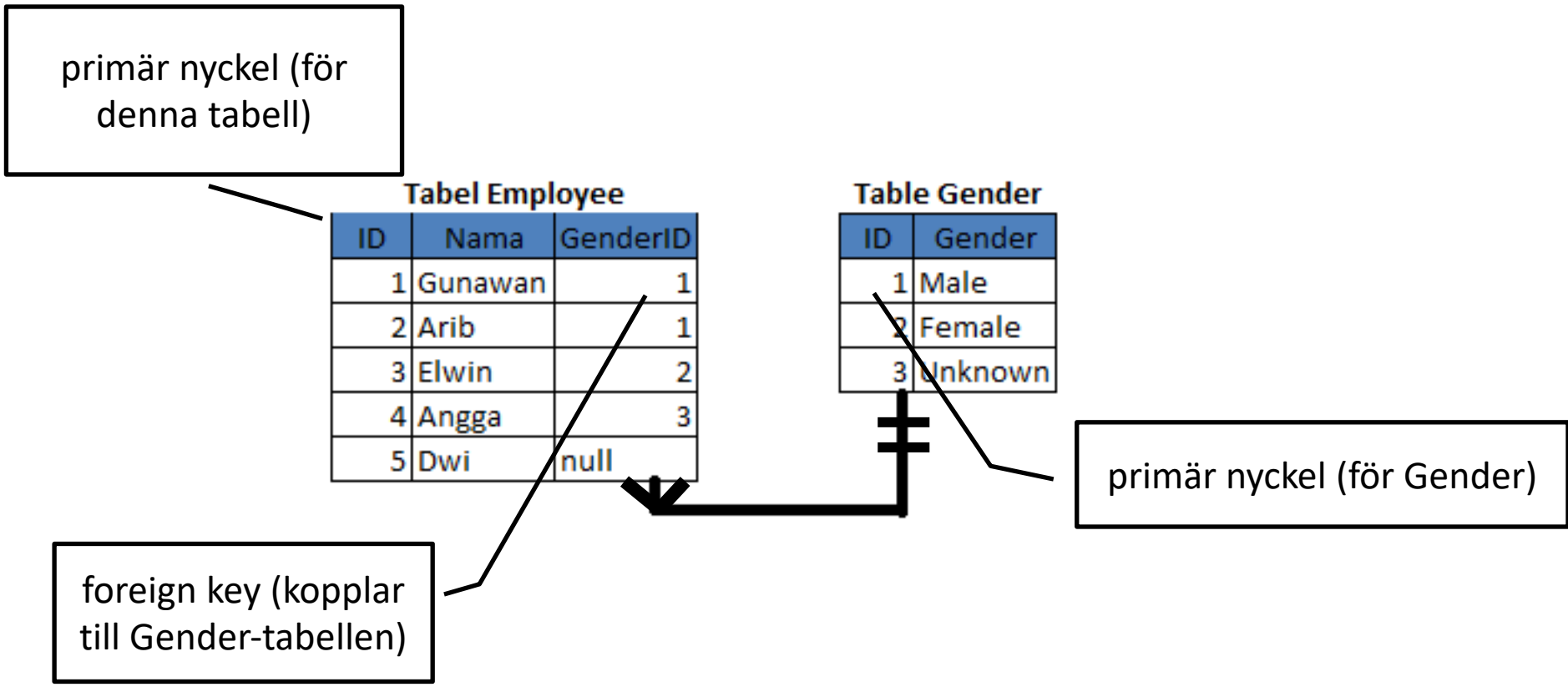
- Låt ena tabellen peka ut unik rad i andra tabellen

Bok #	Bok	Låntagare #
1	Krig och fred	123456
2	Blomskötsel	123456

Låntagare #	Namn
123456	Karl Karlsson
2568946	Sven Svensson

  - Lättare att hålla uppdaterat
    - Slipper söka igenom en gigantisk tabell efter duplikat för varje uppdatering

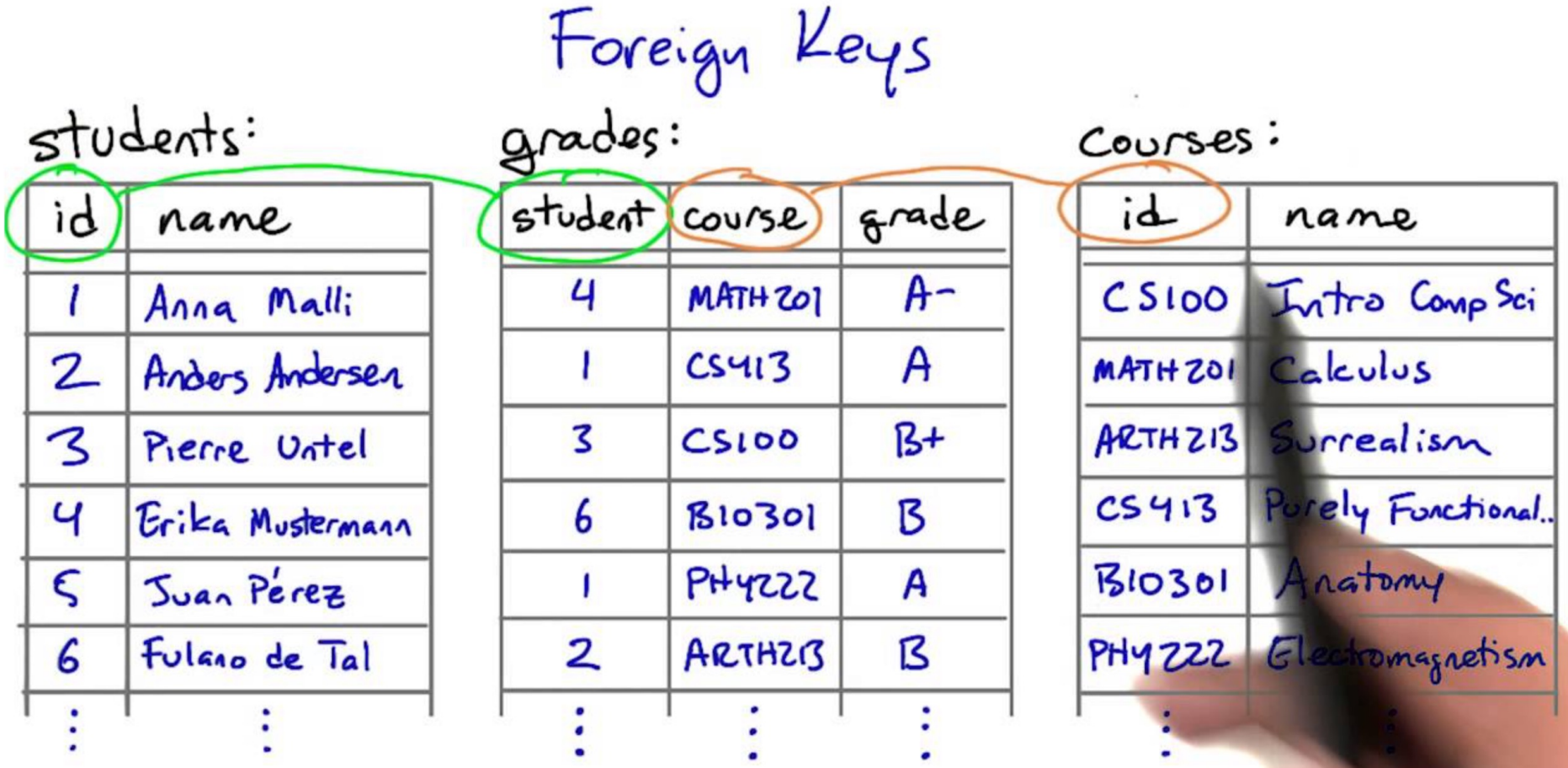
Länka från många till en



Many-to-many relation

associationstabell

Associationstabell



SQL

Structured Query Language

Structured Query Language (SQL)

(Exempel i Microsoft Access)

Vill ha ut dessa två kolumner

- SELECT [Firstname], [Lastname] FROM [Employees] WHERE [Lastname] = "Davolio";

Från denna tabell

Rad(er) som uppfyller detta kriterium

Antag att vi har två tabeller

Employees

#	Name	Department
1	J Devolio	IT
2	K Svensson	Accounting
3	J Jönsson	Accounting
4	P Frilans	null

Managers

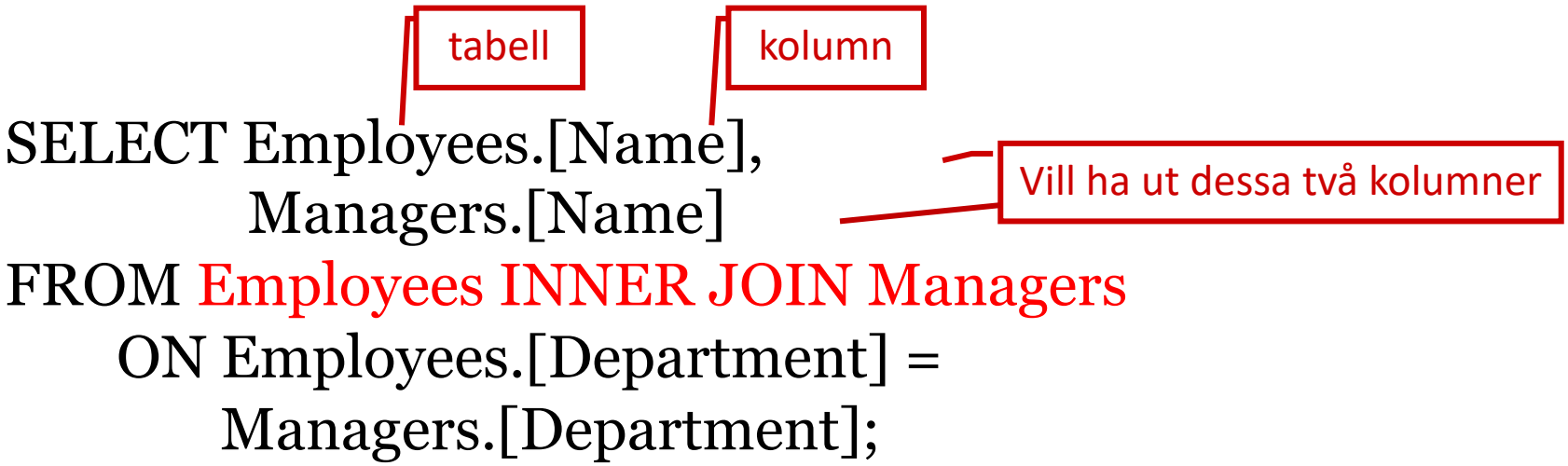
#	Name	Department
6	Karl Karlsson	IT
7	Sven Svensson	Accounting

Vill få ut:

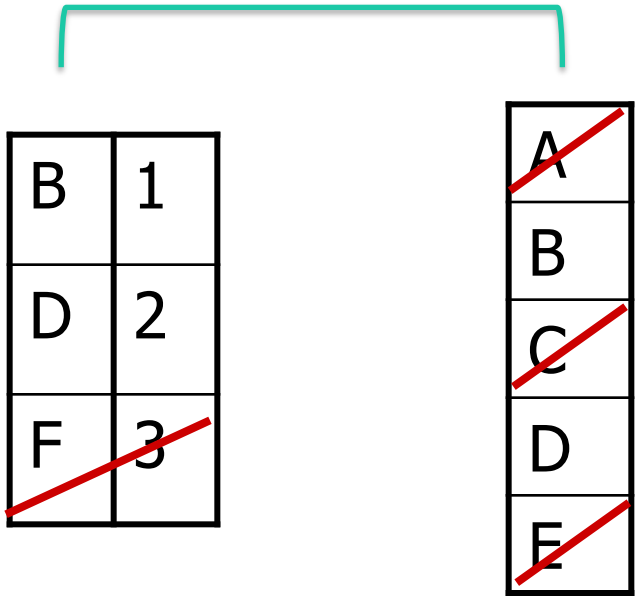
#	Name	Manager
1	J Devolio	Karl Karlsson
2	K Svensson	Sven Svensson
3	J Jönsson	Sven Svensson

Två tabeller kombineras

- Ge mig alla anställda och deras chefer



Inner join (behåll bara matchande)

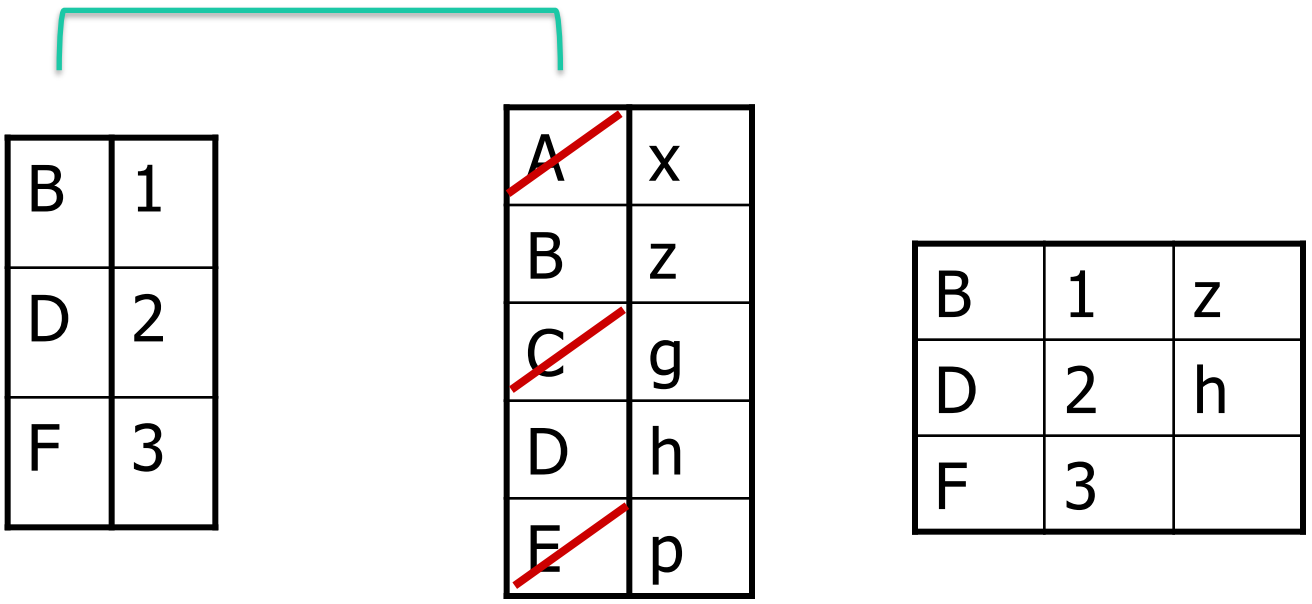


B	1
D	2

A INNER JOIN B

behåll bara matchande rader

Left outer join = behåll alla i vänstra tabellen



B	1
D	2
F	3

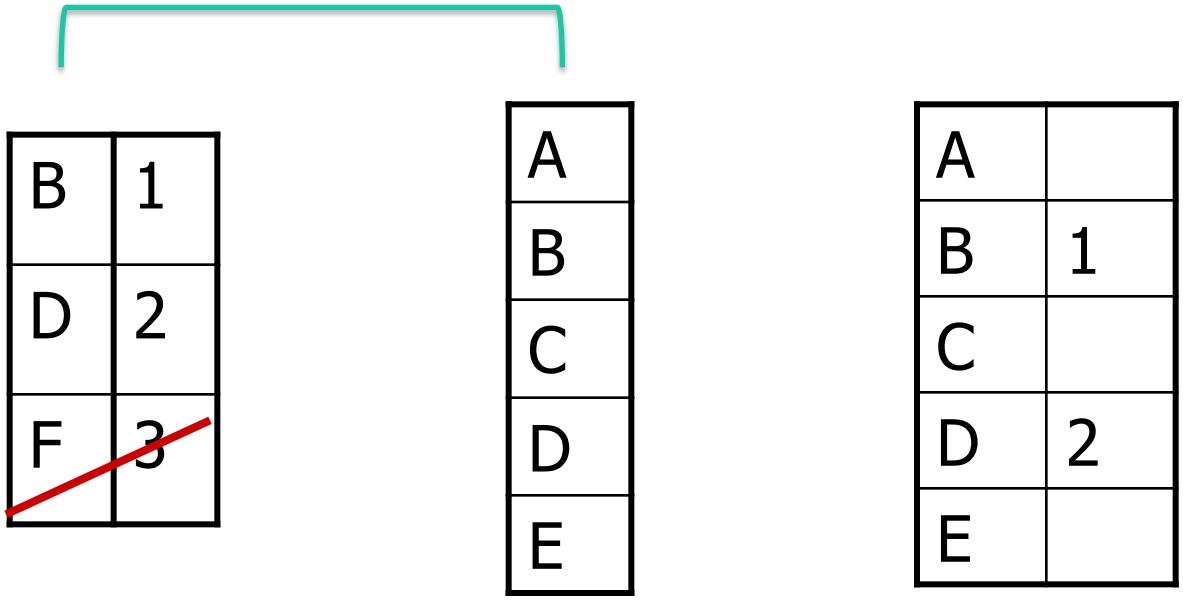
~~A~~	x
B	z
~~C~~	g
D	h
~~E~~	p

B	1	z
D	2	h
F	3	

A LEFT JOIN B

alla rader från A

Right outer join = behåll alla i högra tabellen



A RIGHT JOIN B

behåll alla rader från B

Läs mer på

- SQL
  - https://www.w3schools.com/sql/
- Join
  - https://support.microsoft.com/en-us/office/join-tables-and-queries-3f5838bd-24a0-4832-9bc1-07061a1478f6
  - https://community.qlik.com/thread/39177

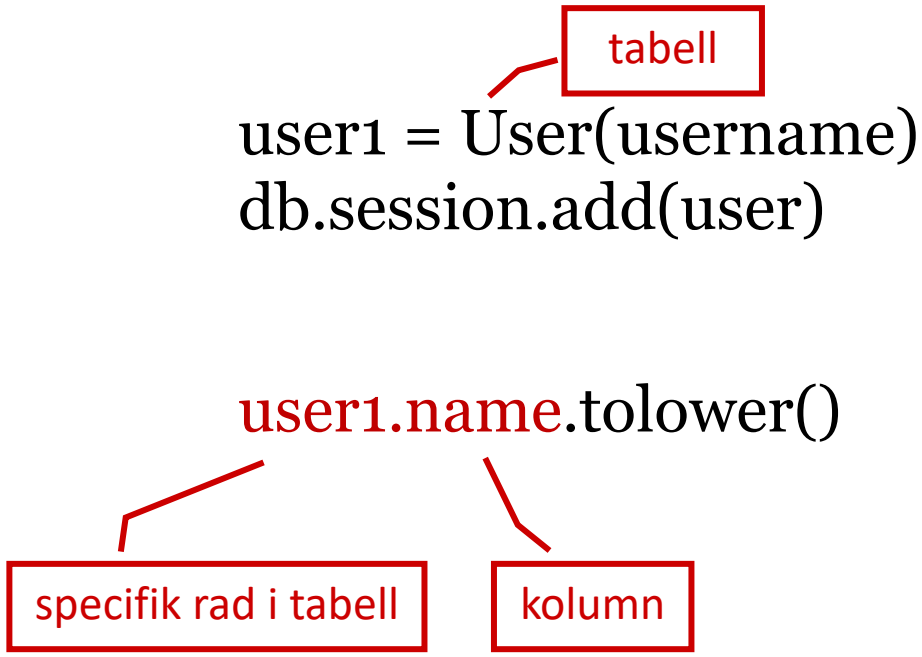
Slippa SQL… SQLAlchemy

SQLAlchemy: Object-Relational Mapper (ORM)

- Vi kan använda python klasser och metoder
- SQLAlchemy översätter
  - Klass till tabell
  - Instans till rad
  - Variabler till kolumner
  - Metoder till queries

SQLAlchemy

- Tillåter vanlig “python”-syntax
  - Dina egna data-repr och python operationer



SQLAlchemy session

- Ändringar i databasen måste commit:as
  - Tex. när man skapat ny instans av User
  - db.session.add(user1)

    // session.dirty är nu True
  - db.session.commit()
    - *Nu* har ändringarna lagts in i databasen

Flask-SQLAlchemy

- Konfigurerar SQLAlchemy, ger oss db-objekt

  db = SQLAlchemy(app)

```
# Tabell för User:
class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(5))
    ...
```

Tre skikt av query-syntax

- Flask-SQLAlchemy
  - Förenklad syntax (query_by, etc.)
- SQLAlchemy
  - Kan använda klasser och vanliga metoder istf "lågnivå" SQL-queries
- SQL
  - Allt översätts till SQL som loggas om man är i debug-mode

Exempel 1

- db.session.query_by(Employees.lastname).
  filter_by(lastname='Davalio').all()

- Employees.query.
  filter(lastname='Davalio').all()

- SELECT lastname
  FROM Employees
  WHERE lastname = "Davolio";

Alla rad(er) som uppfyller detta kriterium

Exempel 2

```
db.session.query(User).
        filter_by(User.name.like('%ed')).order_by(User.id)
```

```
SELECT users.id AS users_id,
        users.name AS users_name,
        users.fullname AS users_fullname,
        users.password AS users_password
FROM users
WHERE users.name LIKE ? ORDER BY users.id
('%ed',)
```

Inner join i Alchemy (exempel)

```
db.session.query(Message).join(User).
                filter(User.name = ‘Kalle’).all()
```

- Skapar en sammanslagen tmp-tabell, med kolumner från Message och User, läser ut de rader som matchar ‘Kalle’

- Se vidare: http://docs.sqlalchemy.org/en/latest/orm/query.html

Dynamic queries

- Om man anger lazy='dynamic' i sin tabell-def:
  - Query exekveras inte utan kan byggas på
  - Kan lägga på ytterligare filter eller operationer på en query

```
db.session.query(Message).join(User).
            filter(User.name = 'Kalle').order(User.name).all()
```

- Exekveras till slut vid en trigger, t.ex. '.all()'

Many-to-many relationer

associationstabell mellan surveys och questions

```
surveys_questions = db.Table(
                'surveys_questions',
                db.Column('surv_id', db.Integer, db.ForeignKey('surveys.id')),
                db.Column('ques_id', db.Integer, db.ForeignKey('questions.id')))


class Survey(db.Model):
    __tablename__ = 'surveys'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True)
    ending = db.Column(db.DateTime)
    messages = db.relationship(…)


class Question(db.Model):
    …
```

Many-to-many relationships

- Läs vidare på:
- http://docs.sqlalchemy.org/en/latest/orm/tutorial.html
  - Klicka på navigationsfältet:
    - Building a Many to Many Relationship

Testning

Requests-anrop

För att kunna köra enklare:
>> ./my_requests.py
lstf
>> python my_requests.py

```
#!/usr/bin/env  python3

import requests

URL_root = 'http://localhost:5000’

resp = requests.post(URL_root + '/messages’, json={'message’: ’...'}
message_id = resp.content.decode(encoding='utf-8')
print('Stored message id: ', message_id)
```

Nackdel med ”requests-testning”

- Databasen blir större och större för varje anrop, för varje varv man testkör sin kod…
- Dålig kontroll över vad som faktiskt finns i databasen om man vill köra ett visst test
  - Så, svårt att veta vad man ska förvänta sig för korrekt svar i ett testfall

- Behöver ha bättre kontroll över kontexten för varje testfall

Enhetstestning (av egen kod, ej helt system)

- Sätt upp fräsch test-databas med testspecifika data
  - Kör ett testfall i taget
  - Automatisk rensning av databasen mellan fallen
- Varje testfall: *sekvens av anrop* som bygger upp databasen
    - T.ex. “lägg i 2 meddelanden i databasen, låt första meddelandet läsas av ‘Kalle, etc.’
  - Slutligt anrop utvärderas m.a.p. databasen som byggdes upp för just detta testfall

Exempel sätta upp DB (unittest I PyCharm)

```
def setUp(self):
    self.db, DB_FILE_PATH = tempfile.mkstemp()
    SQLALCHEMY_DB_URI = 'sqlite:///' + DB_FILE_PATH
    …
    with app.app_context():
        my_database_handler.initialize_db()

def tearDown(self):
    …
```

Exempel testfall (unittest I PyCharm)

Motsv. requests

```
def test_mark_and_retrieve_message(self):
    payload = {'message': 'hej'}
    rv = self.app.post('/messages', data=json.dumps(payload),
                                    content_type='application/json')
    assert rv.status_code == 200
    mess_id = rv.data
    assert len(mess_id) == 8
    …
    rv = self.app.post('/messages/' + mess_id + '/flag/' + user_name1)
    assert …
    …
    rv = self.app.get('/messages/unread/' + user_name2)
    assert b'hej' in rv.data
```

Exempel sätta upp DB (pytest)

```
@pytest.fixture
def client():
    db_fd, db_file_path = tempfile.mkstemp()
    app.config[DATABASE_URI'] = 'sqlite:///' + db_file_path
    app.config['TESTING'] = True
    client = app.test_client()
    with app.app_context():
        data_handler.init_db()
    yield client
    os.close(db_fd)
    os.unlink(db_file_path)
```

Exempel på testfall (pytest)

Motsv. requests

```
def test_save_message(client):
    payload = {'message': 'hello'}
    r = client.post('/messages', json=payload,
                    content_type='application/json')
    message_id = str(r.data.decode(encoding='utf-8'))
    assert len(message_id) == 8
```

Versionshantera *inte* databasen

- Versionshantera all kod
  - Utom dolda filer, log-filer, IDE-inställningar, etc.
- Ej själva databasen
  - Databasen skapas dynamiskt (på den server som ska hantera de riktiga klienterna)

- Databasen ska alltså inte versionshanteras

rita.kovordanyi@liu.se

www.liu.se

LINKÖPINGS UNIVERSITET