Introduction to Databases

Leif Harald Karlsen
`leifhka@ifi.uio.no`

Universitetet i Oslo

12.10.22

Overview of this module

1. Today: Introduction to databases and the relational model
2. Next week: Basic SQL (answer queries)

Curriculum

- The curriculum of this module are the slides from the lectures
- the weekly exercises with solutions given on the semester page
- The mandatory assignment (will be published 19. october)
- The book *SQL Queries For Mere Mortals* should be used as supplement to the slides for more in-depth explanations, and more examples and exercises

Motivation

Why use databases?

Why not just use e.g. Python lists?

```
patients = ["Mary Smith", "Peter Dawson", "Carl Brown"]
```

- Persistence of data
  - Python's data (e.g. lists and variables) is stored in RAM (Random Access Memory)
  - This memory is lost on shutdown/termination
  - We want data to still be there after shutdown/termination
- Scalability of storage size
  - 1 GB of hard disk space much cheaper than 1 GB of RAM
- Separate data from code
  - Python's data is only available to Python's runtime
  - Want data to be usable by multiple applications

All of these problems are solved by the filesystem!

Motivation

Why use databases?

So why not just use files, then?

Python + Files

```
import csv
import os

filea = "a.csv"
fileb = "b.csv"
temp = "temp.csv"
source1 = csv.reader(open(filea,"r"),delimiter=",")
source2 = csv.reader(open(fileb,"r"),delimiter=",")

source2_dict = {}
for row in source2:
    source2_dict[row[0]] = row[1]

with open(temp, "w") as fout:
    csvwriter = csv.writer(fout, delimiter=delim)
    for row in source1:
        if row[1] in source2_dict:
            row[3] = source2_dict[row[1]]
        csvwriter.writerow(row)
os.rename(temp, filea)
```

SQL + Database

```
UPDATE a
   SET c4=b.c2
  FROM b
 WHERE a.c2 = b.c1;
```

Motivation

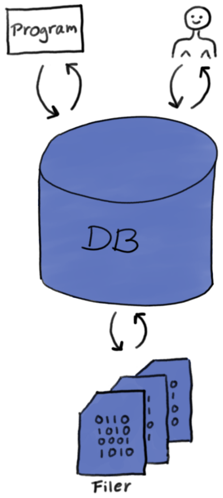
Why use databases?

Why not just use files?

- Convenience of data manipulation
  - Easier to insert, delete and update data
- Query languages
  - For large and complex data, it is easier to state *what* to compute (i.e. what we want to know) rather than how to *compute* it
- Efficiency
  - Database uses advanced techniques to find the most efficient way to execute queries
  - Also uses advanced data structures to store data for efficient retrieval

Motivation
Why use databases?

So why not just files, then?



Database functions as an abstraction layer over the filsystem

- Makes it easier to search and manipulate data
- Easier to specify structure of the data
- More efficient and scalable

Databases

- A database is a program providing easy and efficient access to data
- Different types of databases, focusing on storing different types of data
- Document databases: Stores documents, and can do very efficient search in text
- Key-value stores: Stores pairs of a key and a value
- Graph databases: Stores graphs, i.e., nodes and edges
- Relational databases: Stores tables (or relations) consisting of rows and columns
- We will focus on relational databases, the most used type of database

Relational databases

A (simplified) description of a relational database:

- A relational database is a collection of tables
- A table is also called a relation.
- A table has
  - a name,
  - a collection of columns
  - and a collection of rows (which is the data)
- A column has
  - a name,
  - and a type

Tables/Relations

Example table:

Patient

PatientID (int)	Name (text)	Birthdate (date)	BloodPressure (text)
0	Anna Consuma	1978-10-09	123/75
1	Peter Young	2009-03-01	150/81
2	Carla Smith	1986-06-14	101/53
3	Sam Penny	1961-01-09	127/82
4	John Mill	1989-11-16	147/92
5	Yvonne Potter	1971-04-12	122/74

Rows and Columns

Patient

PatientID (int)	Name (text)	Birthdate (date)	BloodPressure (text)
0	Anna Consuma	1978-10-09	123/75
1	Peter Young	2009-03-01	150/81
2	Carla Smith	1986-06-14	101/53
3	Sam Penny	1961-01-09	127/82
4	John Mill	1989-11-16	147/92
5	Yvonne Potter	1971-04-12	122/74

- Every value within a column must have the same type as the column
  - so the type of a column describes the allowed values in that column
  - E.g. only allowed to put integers into a column having type `int`
- Every row must have the same number of values as there are columns
  - so the columns describes the allowed rows in that table
  - a patient must have `PatientID`, `Name`, `Birthdate`, `BloodPressure`

Why relational databases?

Patient

PatientID (int)	Name (text)	Birthdate (date)	BloodPressure (text)
0	Anna Consuma	1978-10-09	123/75
1	Peter Young	2009-03-01	150/81
2	Carla Smith	1986-06-14	101/53
3	Sam Penny	1961-01-09	127/82
4	John Mill	1989-11-16	147/92
5	Yvonne Potter	1971-04-12	122/74

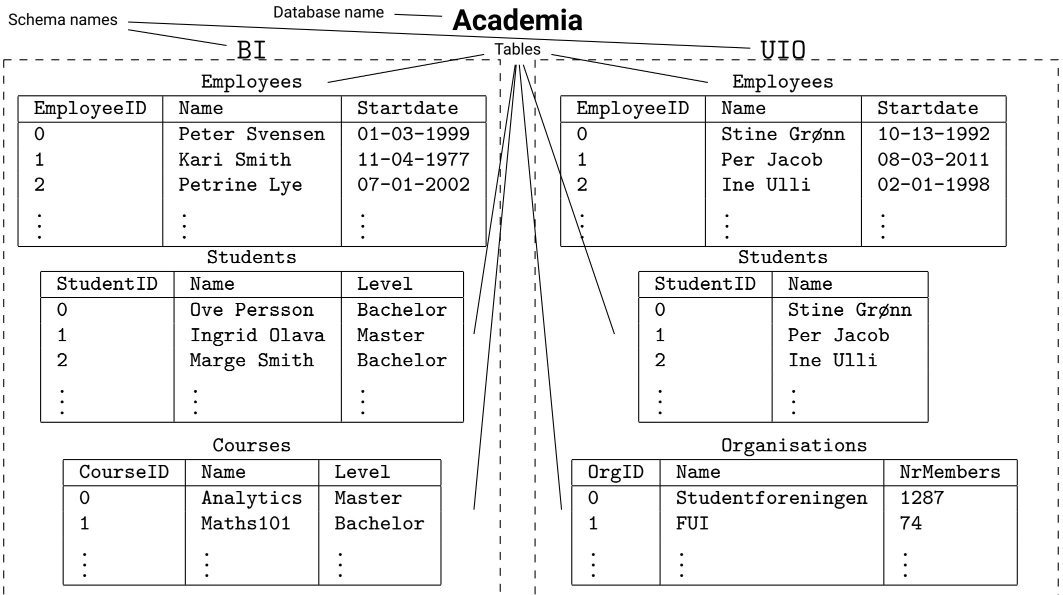
- Almost all data can (naturally) be represented as tables
- Natural format to work with
- Easy to define structure of the data (meta data)
- This rigid structure allows very efficient extraction and manipulation of the data
- Also gives many forms of security
- The most used type of database

Database schema

So

- the table’s columns describe the shape and form of the data
- that is, it is metadata (i.e. data about the data)
- The collection of table names and column names and types are part of the *database schema*
- A database can have multiple such database schemas, and each schema has a name
- Schemas are used to group related tables together (e.g. one schema for tables related to patients, one schema for tables related to hospitals, etc.)

Example Database



BI

Employees

EmployeeID	Name	Startdate
0	Peter Svensen	01-03-1999
1	Kari Smith	11-04-1977
2	Petrine Lye	07-01-2002
⋮	⋮	⋮

Students

StudentID	Name	Level
0	Ove Persson	Bachelor
1	Ingrid Olava	Master
2	Marge Smith	Bachelor
⋮	⋮	⋮

Courses

CourseID	Name	Level
0	Analytics	Master
1	Maths101	Bachelor
⋮	⋮	⋮

UIO

Employees

EmployeeID	Name	Startdate
0	Stine Grønn	10-13-1992
1	Per Jacob	08-03-2011
2	Ine Ulli	02-01-1998
⋮	⋮	⋮

Students

StudentID	Name
0	Stine Grønn
1	Per Jacob
2	Ine Ulli
⋮	⋮

Organisations

OrgID	Name	NrMembers
0	Studentforeningen	1287
1	FUI	74
⋮	⋮	⋮

Relational databases = Spreadsheets?

So, are relational databases just spreadsheets?

No, relational databases has:

- a rigid structure
- query languages for extraction and manipulation of data
- easy access from programming languages (like Python)
- systems for security and control of who has access to the data
- systems that secure the integrity of the data
- support for much larger volumes of data and much more complex structure

Database systems

- A database is really just a collection of data (not a system/program)
- A database management system (DBMS) is

  a system that let users define, create, maintain and control access to data.
- A relational database manegement system (RDBMS) is

  a database management system over relational databases.
- Often use the word “database” for both data, program, and the combaintion of these

Schema violations

The database system will not allow you to insert values violating the database schema.

Thus, the following is not allowed (errors marked in red):

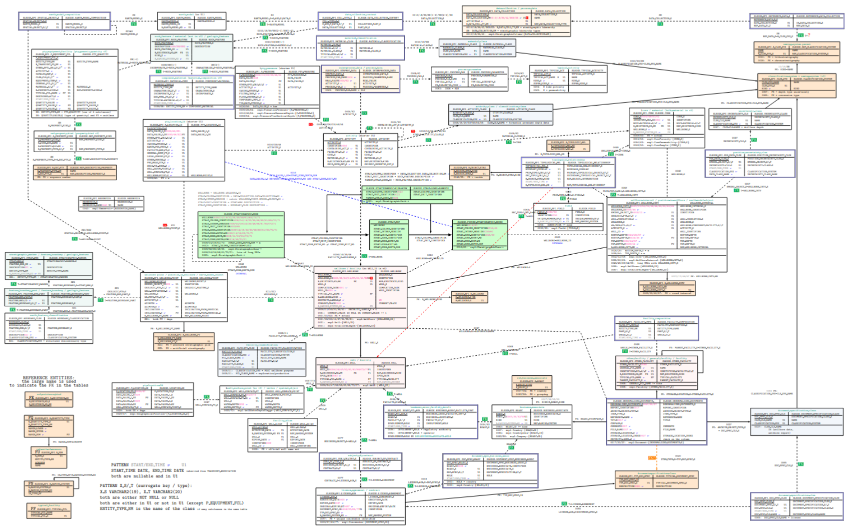
Patient

PatientID (int)	Name (text)	Birthdate (date)	BloodPressure (text)
0	Anna Consuma	1978-10-09	123/75
1	Peter Young	2009-03-01	150/81
2	2	1986-06-14	101/53
3	Sam Penny	long ago	127/82
four	John Mill	1989-11-16	147/92
5 6	Yvonne Potter	1971-04-12	122/74

Database design

- So relational databases store data as tables with a rigid structure
- But how should we represent information as data in tables?
- The structure of the data, i.e. which tables and columns we make, affects how easy it is to use and maintain the data
- Need to have a good *database design*

Complex database schemas


REFERENCE ENTITIES:
the large name is used
to indicate the FK in the tables
PATTERN START/END_TIME U1
START_TIME DATE, END_TIME DATE
both are nullable and in U1
PATTERN X_S/X_T (surrogate key / type):
X_S VARCHAR2(19), X_T VARCHAR2(20)
both are either NOT NULL or NULL
both are either in U1 or not in U1 (except P_EQUIPMENT_FCL)
ENTITY_TYPE_NM is the name of the class

Database design: Blood pressure

Assume we want to keep track of pasient's blood pressure over time. We could then make a table looking like this:

Patient

PatientID	Name	Birthdate	Telephone	BloodPressure	TestTime
0	Anna Consuma	1978-10-09	12345678	123/75	2022-09-23
1	Peter Young	2009-03-01	21679921	150/81	2022-09-20
2	Carla Smith	1986-06-14	98765432	101/53	2022-08-07
3	Sam Penny	1961-01-09	91827364	127/82	2022-09-28
4	John Mill	1989-11-16	56473829	147/92	2022-09-13
5	Yvonne Potter	1971-04-12	91298833	122/74	2022-09-04

Blood pressure: More tests

Patient

PatientID	Name	Birthdate	Telephone	BloodPressure	TestTime
0	Anna Consuma	1978-10-09	12345678	123/75	2022-09-23
1	Peter Young	2009-03-01	21679921	150/81	2022-09-20
2	Carla Smith	1986-06-14	98765432	101/53	2022-08-07
3	Sam Penny	1961-01-09	91827364	127/82	2022-09-28
4	John Mill	1989-11-16	56473829	147/92	2022-09-13
5	Yvonne Potter	1971-04-12	91298833	122/74	2022-09-04
0	Anna Consuma	1978-10-09	12345678	125/73	2022-10-01
1	Peter Young	2009-03-01	21679921	143/80	2022-10-03
4	John Mill	1989-11-16	56473829	146/92	2022-10-03
5	Yvonne Potter	1971-04-12	91298833	124/75	2022-10-04
0	Anna Consuma	1978-10-09	12345678	126/74	2022-10-05
3	Sam Penny	1961-01-09	91827364	126/80	2022-10-08
1	Peter Young	2009-03-01	21679921	141/79	2022-10-11

Problems with bad design

- Difficult to maintain data
  - If a patient changes name or phone number, need to change multiple rows
- Difficult to add data
  - Cannot insert new patient without also inserting blood pressure and testtime
- Duplicate data takes up more disk space and is slower to work with

Blood pressure: (Failed) attempt at better structure

Patient

PatientID	Name	Birthdate	Telephone	BloodPressure
0	Anna Consuma	1978-10-09	12345678	(123/75, 2022-09-23),(125/73, 2022-10-01),...
1	Peter Young	2009-03-01	21679921	(150/81, 2022-09-20),(143/80, 2022-10-03),...
2	Carla Smith	1986-06-14	98765432	(101/53, 2022-08-07)
3	Sam Penny	1961-01-09	91827364	(127/82, 2022-09-28),(126/80, 2022-10-08)
4	John Mill	1989-11-16	56473829	(147/92, 2022-09-13),(146/92, 2022-10-03)
5	Yvonne Potter	1971-04-12	91298833	(122/74, 2022-09-04),(124/75, 2022-10-04)

- Blood pressure values now contained deep inside a single value
- Need to ”parse”/”unwrap” this complex value to get blood pressure values
- Makes working with these values very complex (both for humans and computer)
- Generally: Columns should have simple values!

Blood pressure: Better structure

Patient

PatientID	Name	Birthdate	Telephone
0	Anna Consuma	1978-10-09	12345678
1	Peter Young	2009-03-01	21679921
2	Carla Smith	1986-06-14	98765432
3	Sam Penny	1961-01-09	91827364
4	John Mill	1989-11-16	56473829
5	Yvonne Potter	1971-04-12	91298833

BloodPressure

PatientID	BloodPressure	TestTime
0	123/75	2022-09-23
1	150/81	2022-09-20
2	101/53	2022-08-07
3	127/82	2022-09-28
4	147/92	2022-09-13
5	122/74	2022-09-04
0	125/73	2022-10-01
1	143/80	2022-10-03
4	146/92	2022-10-03
5	124/75	2022-10-04
0	126/74	2022-10-05
3	126/80	2022-10-08
1	141/79	2022-10-11

Students and courses

Want to store information about students, courses and grades:

- For students: Username, name, surename, address...
- For courses: Coursecode, title, description, credits...
- Grades: Which student got which grade in which course

Naive solution: Everything in one table!

Students and courses: Schema

StudentCourse

Username	Name	Surename	Address	Coursecode	Title	Desc.	Credits	Grade
evgenit	Evgenij	Thorstensen	Addr1	IN2090	Databaser	Beskr...	10	B
peternl	Petter	Nilsen	Addr2	IN2090	Databaser	Beskr...	10	A
evgenit	Evgenij	Thorstensen	Addr1	IN2080	Beregn...	Descr...	10	A
leifhka	Leif H.	Karlsen	Addr3	IN2090	Databaser	Beskr...	10	B
leifhka	Leif H.	Karlsen	Addr3	IN3110	Program...	Desc2...	5	C

Insert and delete

StudentCourse

Username	Name	Surename	Address	Coursecode	Title	Desc.	Credits	Grade
evgenit	Evgenij	Thorstensen	Addr1	IN2090	Databaser	Beskr...	10	B
peternl	Petter	Nilsen	Addr2	IN2090	Databaser	Beskr...	10	A
evgenit	Evgenij	Thorstensen	Addr1	IN2080	Beregn...	Descr...	10	A
leifhka	Leif H.	Karlsen	Addr3	IN2090	Databaser	Beskr...	10	B
leifhka	Leif H.	Karlsen	Addr3	IN3110	Program...	Desc2...	5	C
				IN9999	Quantum	Beskr3	10	
abcdef	Aber C.	Deflan	Addr4					

Data duplication makes it more difficult to insert and update data:

- Need to insert all the info about student and course, even if only want to insert a new grade
- Impossible to insert a new student, without also inserting a course
- Updates must be performed consistently and update all duplicates

Anomalier: Sletting

StudentCourse

Username	Name	Surename	Address	Coursecode	Title	Desc.	Credits	Grade
evgenit	Evgenij	Thorstensen	Addr1	IN2090	Databaser	Beskr...	10	B
peternl	Petter	Nilsen	Addr2	IN2090	Databaser	Beskr...	10	A
evgenit	Evgenij	Thorstensen	Addr1	IN2080	Beregn...	Descr...	10	A
leifhka	Leif H.	Karlsen	Addr3	IN2090	Databaser	Beskr...	10	B
leifhka	Leif H.	Karlsen	Addr3	IN3110	Program...	Beskr2...	5	C

- Deleting a course may delete a student
- Deleting a student may delete a course
- Difficult to fix this with this structure

Fix the structure

Username	Name	Surename	Address	Coursecode	Title	Desc.	Credits	Grade
evgenit	Evgenij	Thorstensen	Addr1	IN2090	Databaser	Beskr...	10	B
peternl	Petter	Nilsen	Addr2	IN2090	Databaser	Beskr...	10	A
evgenit	Evgenij	Thorstensen	Addr1	IN2080	Beregn...	Descr...	10	A
leifhka	Leif H.	Karlsen	Addr3	IN2090	Databaser	Beskr...	10	B
leifhka	Leif H.	Karlsen	Addr3	IN3110	Program...	Beskr2...	5	C

Student

Username	Name	Surename	Address
evgenit	Evgenij	Thorstensen	Addr1
peternl	Petter	Nilsen	Addr2
leifhka	Leif H.	Karlsen	Addr3

Grade

Username	Coursecode	Grade
evgenit	IN2090	B
peternl	IN2090	A
evgenit	IN2080	B
leifhka	IN2090	B
leifhka	IN3110	C

Course

Coursecode	Title	Desc.	Credits
IN2090	Databaser	Beskr...	10
IN2080	Beregn...	Descr...	10
IN3110	Program...	Beskr2...	5

- Note: Same columns and same values!
- Good structure: Separate table for students, courses and grades

Design Principles

Rules of thumb for database design:

- One table per type of thing (patient, student, course)
  - One column per attribute/property (name, telephone, title, etc.)
- One table per relationship (grade)
- One table per multi-valued property (blood pressure)

Not only one good structure!

- There can be many good ways of structuring the same information into tables.
- The following two tables on *life expectancy in Norway* contain the same information without data duplication, but is structured differently:

Year	Men	Women
2017	80.9	84.3
2018	81.0	84.5
2019	81.2	84.7
2020	81.5	84.9

Gender	Year	LE
men	2017	80.9
men	2018	81.0
men	2019	81.2
men	2020	81.5
women	2017	84.3
women	2018	84.5
women	2019	84.7
women	2020	84.9

Keys: Managing identity

- To describe something, we need to have a unique way of referencing it
  - E.g. if I say ”Peter has blood pressure 120/80” and there are two patients with name ”Peter”, we do not know which ”Peter” we talk about
- Luckily, most things in the real world have this
  - People: Personnummer (combination of fødselsnummer and birthdate)
  - Buildings: Address
  - Cars: License plates
  - Products: Barcode
  - Students: Username
  - ...
- In a database, every type of thing (students, courses, patients) needs to have a column (or combination of columns) that is unique for that type of thing (`Brukernavn`, `Coursecode`, `PatientID`)
- These columns are called *keys* or *primary keys*

Foreign keys

- When one table references another, we do this via such keys
- E.g. in `BloodPressure` we used `PatientID` to reference a patient and in `Grade` we used `Username` to reference students and `Coursecode` to reference courses
- Such references are known as *foreign keys*

Example of common problem: Data integration and communication

- When organizations/companies merge, they need to merge their data
- Similarly when communicating data between organizations
- Data needs to be of the same format/same structure
- Values must denote the same thing
- Merging data into a common format/structure known as *data integration*

Example of common problem: Data integration and communication

- Data integration is a very difficult problem, as the different organizations typically:
  - Have large and complex database schemas
  - Use different keys for the same type of things
  - Use different structure for same type of information
  - Have lots of applications and systems using their data the way it is stored
  - Use different database management systems that are not compatible
- Standardization helps solving these problems

Year	Men	Women
2017	80.9	84.3
2018	81.0	84.5
2019	81.2	84.7
2020	81.5	84.9

Gender	Year	LE
men	2017	80.9
men	2018	81.0
men	2019	81.2
men	2020	81.5
women	2017	84.3
women	2018	84.5
women	2019	84.7
women	2020	84.9