University of Basel

Prof. Dr. Heiko Schuldt
David Lengweiler, M. Sc.
Martin Vahlensieck, M. Sc.

Introduction to Databases — Autumn 2023

Exercise 5

(Theory) Hand-in: 17.12.2023 (ADAM, 23:59)
(Practical) Hand-in: 22.12.2023 (during Exercise)

Solving the Exercises: The exercises can be solved in small groups of a maximum of two people. Use the notations introduced in the lecture. The DMI plagiarism guidelines apply for this lecture.

Submission Information: Please upload all (Theory) deliverables BEFORE the deadline to ADAM as **a single PDF** and all (Practical) as **separate text files** using the team hand-in feature. Solutions that are handed in too late cannot be considered. For practical exercises upload the deliverables to ADAM and present them to one of the assistants/tutors during the exercises, both is required to recieve the points!

Task (Practical) 1: SQLite (1 points)

As mentioned throughout the lecture, there are many database systems. Most of these (including PostgreSQL) work in a client-server model. This means that the database system is running by itself waiting for clients to connect to it and send commands.

In contrast to this are embedded databases. These work by including the code on how to read and write database files within the application (via a package, or a library). This means no setup of a database system is required. A very popular example of such an embedded database system is SQLite.

Some programming languages, like Python, have an SQLite adapter in their standard library. Therefore we recommend you use Python for these tasks. Just do `import sqlite3` and you are good to go. As both the SQLite driver and the PostgreSQL driver follow the Python DBI 2.0 interface standard the only difference is the line where you connect to the database:

```
con = psycopg2.connect(host="localhost", user="demo",
    password="demo", port=54321)
```

```
con = sqlite3.connect('example.db')
```

Here `example.db` is the name of the file in which the database is stored.

There is also a SQLite shell to interact with database files by hand. Download a copy from the official website for your operating system. This shell is in fact nothing other than an application, which has the database code included.

Try to import the `ex1.sql` dump from the first exercise into a SQLite database. This leads to some problems, why is this the case? *(You do not have to fix the problems!)*

Hand-In: Show the working shell to one of the assistants/tutors.

Task (Practical) 2: Reading data from CSV (4 points)

In this task you will do a mini analysis using SQLite, which you set up in the first task. This involves downloading some data, reading it into the database, performing analytical queries, plotting the result and finally creating a dump of the database.

Download the dataset[1] with the last names of the population in Basel-Stadt. This file is the *CSV* (comma-separated values) format. This format is quite simple with rows corresponding to lines and the columns separated by some separator. While the name would indicate this is a comma (`,`) it is in this specific case a semi-colon (`;`).

Look at the first few lines. There are quite a few redundancies in the file, such as the date and then a column just for the year of that date. Popular names will be saved many times over as well, so you will optimize this by creating two tables with the following relational schema:

names(id, name)

counts(id, date, count)

Create the schema with the primary and foreign keys. Also add constraints to ensure the name is both unique and not empty, and that the count is positive.

Hint: If you do not specify the value of an integer primary key on insert, SQLite will choose a free value.

Hint: Foreign key constraints are off by default in SQLite. Use `pragma foreign_keys=on` to turn them on.

Hint: You need to run `con.commit()` to persist the inserted data to disk.

Insert the data from the CSV. Compare the size of the database with the size of the CSV. What do you notice? Do you have an explanation?

Answer the following questions with one SQL statement each. *Hint:* “übrige” means “other”.

(1) How often does a given last name appear over the years? Use your last name as a placeholder.

(2) Create a view that recreates the original CSV table. To retrieve the year from the date you can use the `strftime` function like this: `strftime('%Y', date)`

(3) How many times does each count appear, taken over all years? Only retrieve the

[1] `https://opendata.swiss/en/dataset/nachnamen-der-baselstadtischen-bevolkerung/resource/da4ec2fd-d32b-4cbc-9ee6-a7d905d67389`

counts that appear more than five times.

Plot the results of the last question with the method of your choice. If you choose Excel, there is an easy shorthand in the SQLite shell: Just type `.excel` on a line by itself and after that the query. Voilà — Excel should open with the results. How this works is that SQLite saves the output of the query as a CSV file and then opens it with Excel. You can also export a CSV file by itself using the `.csv filename` command. There are many more options, see the `.help` command.

One important task when handling data is backing it up. With SQLite this is simple as you can copy the file. Be aware that this can be quite dangerous if the file is copied while it is used by another application: the resulting file can be corrupt! A safer option is the `.dump` command. Use this to create a dump of your database.

Hand-In: Hand in your script to read the data, the queries you used to answer the questions and the SQL dump to ADAM. Show the plot to one of the assistants/tutors.

Task (Practical) 3: NoSQL (7 points)

In this exercise you will be introduced to a key-value store. As a practical implementation, we choose Redis. As the feature set of Redis goes up and beyond that of a traditional key-value store, we will only use a small subset of commands.

Again, the easiest way to get a Redis instance is with Docker:

```
docker run --name introdb-redis -p 127.0.0.1:6379:6379 -d \
     redis/redis-stack-server:latest
```

But as always you are free to set it up any other way.

To interact with the store, either use the command line like this

```
docker exec -it introdb-redis redis-cli
```

or use DataGrip.

You will also need to be able to interact with it in a programming language. For Python follow the instructions of the second exercise substituting the `psycopg2-binary` package with the `redis` package.

Connect in Python like this:

```
  import redis

  client = redis.Redis(host='127.0.0.1', port=6379,
      decode_responses=True)
```

You are only allowed to use commands/operations presented in the lecture, which are the following:

Lecture	Redis Cli	Python
get(key)	GET key	client.get(key)
put(key, value)	SET key value	client.set(key, value)
delete(key)	DELETE key	client.delete(key)

Insert the data from the CSV. You must enforce the same constraints as in task 2.

Now write a second program using only the database (not the CSV), that answers the following questions:

(1) Have your program prompt the user for a last name and print how often this name appears over the years.

(2) Recreate the CSV file from the data stored in Redis.

(3) How many times does each count appear, taken over all years? Only retrieve the counts that appear more than five times.

Hand-In: Upload both programs one for inserting the data and one for answering the questions to ADAM.

Task (Theory) 4: Integrity (3 points)

You are tasked with designing a table which stores budget request for an accounting firm. For this you are given the following boilerplate for a CREATE TABLE statement:

```
CREATE TABLE request (
    title VARCHAR(255) [...],      NOT NULL   -> weil Primary key teil
    ~~applicant VARCHAR(255)~~ [...], NOT NULL
    ~~reviewer1 VARCHAR(255)~~ [...], NOT NULL
    ~~reviewer2 VARCHAR(255)~~ [...],
    amount INTEGER [...],         NOT NULL
    [...]
);  PRIMARY KEY (applicant, title)
    Foreign Key applicant reviewer 1,2 references people (id) on delete set Null
```

Additionally, you are also given the following table, which holds all people.

Constraint amountLimit Check (amount between 200 and 500000)

```
CREATE TABLE people (
    f_name VARCHAR(255),
    l_name VARCHAR(255),
    id VARCHAR(255) NOT NULL,
    PRIMARY KEY(id) true
);
```

true

check (amount ≤ 50'000 OR reviewer 2 IS NOT NULL)

Complete the CREATE TABLE statement request and ensure that the following constraints hold:

check (applicant <> reviewer 1 and applicant <> reviewer 2)

check (reviewer2 is null or reviwer 2 <> reviewer 1)

- Each request requires an `applicant` and `amount`.
- A request is identified by its `applicant` and its `title`.
- `reviewer1`, `reviewer2` and `applicant` are people.
- The requested `amount` should be between 200 and 500'000.
- The `title` cannot be empty.
- Each row requires at least a `reviewer1`, if the amount is larger than 50'000 also a `reviewer2` is required.
- The applicant can not also be a `reviewer` for the same request.
- If there are 2 `reviewers` they must be distinct.

Task (Theory) 5: View Update (2 points)

You are given an **empty** table `messurements(id, x)`. You now create a view which uses this table as follows: *`SIN` applies the sinus function to the enclosed value.*

```
CREATE VIEW(id, y) sinus AS
SELECT id, SIN(x) FROM messurements
```

You are now executing the following `INSERT`s on `sinus`, which both fail:

```
INSERT INTO sinus(id,y) VALUES(1, 0.5);
INSERT INTO sinus(id,y) VALUES(2, 2);
```

↳ sin(x) must be $-1 \leq y \leq 1$ 1

To allow for at least on of them to execute successfully you decide to add a check constraint on messurements. Answer the following questions:

(1) Why did the initial `INSERT`s fail?
(2) What are possible check constraint on `messurements`, which allow for at least one of the `INSERT`s to succeed.
(3) How would whe the resulting row(s) in `messurements` look like.

2) Create table measurements

---

constraint .... check (x is not null and $0 \leq x$ and $x \leq 2 \cdot$pi())

3) measurements

id	x

or

id	x
1	0.5

?

Task (Theory) 6: NoSQL (3 points)

For this exercise you are presented with multiple data management scenarios and possible databases/model to deal with them. Pick the most suitable Database type (*Relational Database*, *Key-Value Store*, *Column Store*, *Document Database*, *Graph Database*), explain your decision shortly (1-2 sentences) and provide a rough approach how you would store the data in the database (1-3 sentences). Each database type has to be used at least once.

(1) Infrastructure of air, railway and vehicle network.
(2) Bookkeeping system which havily relies on aggregating the stored values.
(3) List of Students, Exercises and how many points they achieved for each of them.
(4) Log files with different content collected from network switches.
(5) Website cache, which stores the results of exepensive payloads.
(6) Museum storing metadata about its collection of art, with attributes varying between exhibits.

1) Graph : Network -> Nodes Slide 6-29

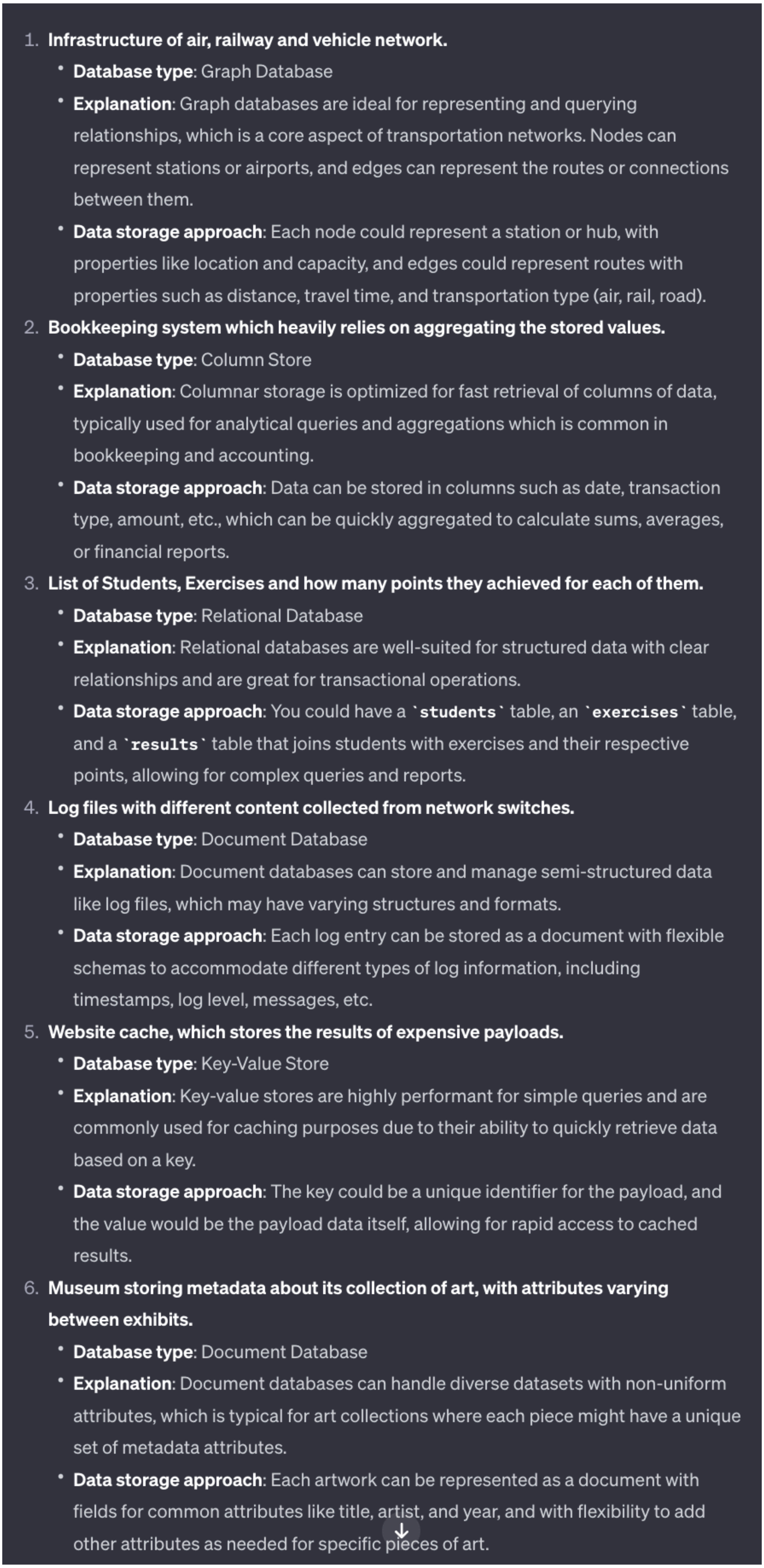
2) Column Store: fast access to many colums 6-18

3) Relational

4) Document Store, many documents

5) Key Value Store

6) Documents



1. **Infrastructure of air, railway and vehicle network.**
   - **Database type**: Graph Database
   - **Explanation**: Graph databases are ideal for representing and querying relationships, which is a core aspect of transportation networks. Nodes can represent stations or airports, and edges can represent the routes or connections between them.
   - **Data storage approach**: Each node could represent a station or hub, with properties like location and capacity, and edges could represent routes with properties such as distance, travel time, and transportation type (air, rail, road).
2. **Bookkeeping system which heavily relies on aggregating the stored values.**
   - **Database type**: Column Store
   - **Explanation**: Columnar storage is optimized for fast retrieval of columns of data, typically used for analytical queries and aggregations which is common in bookkeeping and accounting.
   - **Data storage approach**: Data can be stored in columns such as date, transaction type, amount, etc., which can be quickly aggregated to calculate sums, averages, or financial reports.
3. **List of Students, Exercises and how many points they achieved for each of them.**
   - **Database type**: Relational Database
   - **Explanation**: Relational databases are well-suited for structured data with clear relationships and are great for transactional operations.
   - **Data storage approach**: You could have a **`students`** table, an **`exercises`** table, and a **`results`** table that joins students with exercises and their respective points, allowing for complex queries and reports.
4. **Log files with different content collected from network switches.**
   - **Database type**: Document Database
   - **Explanation**: Document databases can store and manage semi-structured data like log files, which may have varying structures and formats.
   - **Data storage approach**: Each log entry can be stored as a document with flexible schemas to accommodate different types of log information, including timestamps, log level, messages, etc.
5. **Website cache, which stores the results of expensive payloads.**
   - **Database type**: Key-Value Store
   - **Explanation**: Key-value stores are highly performant for simple queries and are commonly used for caching purposes due to their ability to quickly retrieve data based on a key.
   - **Data storage approach**: The key could be a unique identifier for the payload, and the value would be the payload data itself, allowing for rapid access to cached results.
6. **Museum storing metadata about its collection of art, with attributes varying between exhibits.**
   - **Database type**: Document Database
   - **Explanation**: Document databases can handle diverse datasets with non-uniform attributes, which is typical for art collections where each piece might have a unique set of metadata attributes.
   - **Data storage approach**: Each artwork can be represented as a document with fields for common attributes like title, artist, and year, and with flexibility to add other attributes as needed for specific pieces of art.