Philipps-University Marburg
Department Mathematics and Computer Science
Working Group Database Systems

Wintersemester 2025
24.10.2025

Prof. Dr. Bernhard Seeger
Manuel Oed

Exercises for the lecture
NoSQL-Database Systems

Submission: 07. 11. 2025,
no later than 10:00 Uhr
via ILIAS

Exercise 1

Note: It is mandatory to submit your solutions in groups of 3 people! In addition to the lecture and the tutorial, you can use the forum in ILIAS to find a group. For the submission, create a team with the group members in ILIAS. Then combine all your files into a single ZIP file and upload it to the exercise unit *Sheet 01* before the deadline. Please use the "Upload file"/"Datei hochladen" function and not "Upload multiple files as a ZIP archive"/"Mehrere Dateien als Zip-Archiv hochladen" to submit your exercise sheet. Note that the first task of this sheet is mandatory (counts as one of the tasks, of which you must get at least 40% of the overall points). The second task is optional.

Task 1.1: Redis as Cache (5+5+5+5+5+5+5) (35 Punkte)

In this exercise, we will use an H2-Store for persisting data about customers and products in a shop. Additionally, we use Redis as a cache layer for fast access to the most recent data. In the files `H2Example.java` and `JedisBasics.java`, you will find a simple introduction to H2 and the Jedis library, which you should use as an orientation. To use Jedis, you should install Redis and run a local instance.

a) Create a class `H2Shop` that implements the `Shop` interface. This class only uses a `H2` database to persist the data.

b) Create a class `RedisCacheShop` that implements the `Shop` interface. Make sure that all transactions are also persisted in an `H2` Database. Implement the method `addProductToShop` such that all products are additionally stored in Redis.

c) Implement the `buyProduct` method so that the last 5 products purchased by each customer are stored in Redis.

d) Implement the `hasBought` method so that Redis is queried first. If there are no results there, query the `H2` database.

e) Similar to **d)**, implement the `getProduct` method so that Redis is queried first. If no results are found there, query the `H2` database.

f) Check your implementation by creating a `main` method in a class `ShopSimulation`. Test the `getProduct` method with **(i)** an existing `productId` in Redis and **(ii)** with a key that is not stored in either Redis or `H2`.

g) Complete the `main` method from the `ShopSimulation` class and create several customers and products. Have several customers buy more than 5 different products. Use `hasBought` to check at least the following 3 cases:

- A (Customer, Product) pair that is in the cache.
- A (Customer, Product) pair that is in `H2` but not in the cache.
- A (Customer, Product) pair where the customer did not purchase the product.

Task 1.2: Hashing by Hand **(0 Punkte)**

Hash tables are a popular implementation option for a key-value store. In this exercise, you will review various hashing methods and execute them manually. After each insertion, draw the status of the hash table. Submit your results as a PDF file.

Insert the following data records with the keys: 55, 70, 125, 3, 8, 133, 16, 21, 29 in this order into a hash table of length 13 using different strategies. The value of each key is the textual representation of the number in English, i.e., for 55, the key-value pair <55, fifty-five> is inserted.

a) Direct chaining with hash function $h(k) = k \mod 13$

b) Alternating square probing with hash function $h(k) = k \mod 13$

Handling collisions via alternating square probing uses a probing sequence

$$h(k,i) = h(k) + (-1)^{i+1} \cdot i^2 \text{ mod } 13, \quad i = 1,2,3,\dots.$$

More precisely, when inserting an element k, an attempt is first made to insert the element into the hashtable at position $h(k,0) = h(k)$ mod 13. If this bucket is already occupied by some other key, we try the position $h(k,1) = h(k)+1$ mod 13. If the bucket is again occupied, we try the position $h(k,2) = h(k) - 2^2$ mod 13, etc, until we find a bucket that is free.

c) Cuckoo hashing

In cuckoo hashing, two hash tables (H and H') are used, each with its own hash function ($h(k) = k \mod 13$ and $h'(k) = \lfloor \frac{k}{13} \rfloor \mod 13$). When inserting an element i, an attempt is first made to insert the element into H. If the space is already occupied by an element j, an attempt is made to insert the element into H'. If this space is also occupied, i replaces the element j in H, and an attempt is made to insert j into H'. If this space is occupied by an element k, j replaces the element k in H', and an attempt is made to insert k into H. This alternating replacement continues until a free space is found. This can potentially result in an infinite loop.