Question 1: Denormalization and rewriting (1 P.)

a) For this question, we will take another look at the schema from sheet 2:

Store:	[SID, City, EID]
Employee:	[EID, Name, Salary, SID]
Article:	[AID, Name, Producer, Price, ProductionDate]
Inventory:	[AID, SID, Count]
Invoice:	[IID, SID, Customer]
Item:	[IID, SID, AID, Count]

How can you denormalize this schema, to answer the following queries efficiently? Which downsides will this have?
Describe the normal forms of the original and changed schema in your answer. Also note which additional steps have to be taken to keep the data consistent.

Required submission: Denormalized table schema; Explanation and downsides of denormalization; Query after denormalization; Explanation on how to keep data consistent; Normal form of original table schema; Normal form of new table schema.

- Q7: All employees per city
- Q8: The price of all items in given invoice IID = 5.

b) Rewrite the following queries to more efficient queries, returning the same result set. Consider the conditions noted for each of the queries.

Required submission: Simplified query; Explanation why changes where made.

i) Database as specified above

```
SELECT DISTINCT *
FROM Employee
```

ii) There is no index.

```
SELECT MAX(E.Salary)
FROM Employee E
GROUP BY E.EID
HAVING E.EID=2002
```

iii) Attribute `SID` in `Employee` is a foreign key, referencing `SID` in `Store`.

```
SELECT S.SID
FROM Store S, Employee E
WHERE  S.SID = E.SID
```

Question 2: Min-Hashing (1 P.)

Implement Min-Hashing in a language of your choice. Your program has to take two text files as input and parse them into sets of words. The program has to calculate and display:

- Jaccard coefficient between the two texts.
- The similarity estimated by Min-Hashing with k different hash functions (for each $k \in [1, 6]$)
- The similarity estimated by Min-Hashing with k different min values of one hash functions (for each $k \in [1, 6]$)

You can use the template in OLAT, which already parses the files and provides 6 hash functions. Submit the code and the output of your program when executed with the two data files provided in OLAT. If you do not use the template, also submit instructions on how to compile and execute your program.

Required submission: Source Code in separate file; Output after executing code with provided data files; (Compile/Execute instructions if not using template).

Question 3: Quadtrees (1 P.)

a) Give 8 points, with coordinates in the range $[0, 100]$, and sort them into two insertion orders. The first should have less empty leaf nodes if inserted into a quadtree than if they were inserted into a PR quadtree. The second one the other way around.

   Submit the points, the two insertion orders and draw the (PR) quadtrees. You can either draw them as tree or as the grid visualization.

   Required submission: 8 named (x,y) coordinates; Two insertion orders; Drawn PR quadtree and quadtree per insertion order.

b) Assuming we have the whole dataset present upon insertion, provide a pseudo code that optimizes the creation of the quadtree, such that the height of the resulting quadtree is minimized. Note that before the bulk-loading preprocessing of the data is allowed.

   Required submission: Pseudo code; Pseudo code explanation.

c) Assuming we are given uniformly distributed point data, what is the probability that a node at depth k contains a particular point in a PR Quad tree? Additionally, for a collection of v points, calculate the probability that none of the points lies in a given cell at depth k?

   Required submission: Formula to calculate probability;

Question 4: kd tree (1 P.)

a) Insert the following values, in provided order, into an empty kd tree, with $k = 3$:

1. (72, 22, 42)
2. (69, 34, 90)
3. (27, 41, 53)
4. (9, 84, 75)
5. (10, 70, 7)
6. (63, 26, 45)
7. (10, 83, 11)
8. (48, 85, 10)
9. (95, 52, 46)
10. (22, 53, 41)
11. (49, 22, 27)
12. (28, 81, 21)

Draw the result as a tree, like shown in the lecture. Note which values you used for the split.

Required submission: Visualized kd tree; Split dimension per node.

b) Adaptive kd tree

Provide an algorithm, in pseudo code, that takes a list of coordinates in k dimensions and builds a kd tree. This tree may have a maximum of ten data points in its leaves. The dimension used for the split should be the one with the largest variance. The value used for the split, should be the average of the split dimension of all relevant coordinates.

Required submission: Pseudo code; Pseudo code explanation.