

Otto-von-Guericke University Magdeburg

Faculty of Computer Science



Bachelor Thesis

Slide-recommendation for improving students SQL solutions using natural language processing

Author:

Vincent Toulouse

July 30, 2021

Advisors:

Prof. Dr. rer. nat. habil. Gunter Saake

Department of Technical and Business Information Systems

Otto-von-Guericke Universität

M.Sc. Chukwuka Victor Obionwu

Department of Technical and Business Information Systems

Otto-von-Guericke University Magdeburg

Dr. Ing. David Broneske

Deutsches Zentrum für Hochschul- und Wissenschaftsforschung

Toulouse, Vincent:

Slide-recommendation for improving students SQL solutions using natural language processing

Bachelor Thesis, Otto-von-Guericke University Magdeburg, 2021.

Abstract

Virtually every computer science student around the globe attends a database course as part of their study program. All of those courses will teach students to understand and use relational databases since relational databases are the most used type of databases in the last decades. The popular query language SQL is used to interact with relational databases and as such is part of most curricula. The learning experience of understanding SQL can be too steep for students to overcome on their own and some will eventually fail the course. The database group of the Otto-von-Guericke-University developed the online-tool SQLValidator in order to support the students in overcoming the challenges of learning SQL. SQL exercises and quizzes are presented online with automated feedback for students' solutions. The feedback a student receives is essential for their understanding of SQL. In this thesis, we provide a method to automatically improve the feedback for students by mapping SQL exercises with their corresponding lecture materials. The SQL related content of both exercises and slides of the lecture is analyzed based on the cosine similarity using a tf*idf weighting scheme. This way students that have problems successfully solving an exercise will receive a hint as to which slides of the lecture are helpful for the particular exercise. Further optimizations on this mapping increase the effectiveness of linking the lecture's materials with SQL exercises by using the cosine similarity.

Acknowledgments

I would like to thank Prof. Dr. rer. nat. habil. Gunter Saake for providing me with the opportunity to write my bachelor thesis in the DBSE research group.

I am very grateful for the guidance of my advisors Dr.-Ing. David Broneske and M.Sc. Chukwuka Victor Obionwu.

Lastly, I would like to thank my family and friends for their support.

Contents

List of Figures	ix
List of Tables	xi
List of Code Listings	xiii
1 Introduction	1
2 Background	3
2.1 Tf*idf	3
2.2 Term frequency	4
2.2.1 Maximum tf normalization	4
2.2.2 Augmented normalized term frequency	5
2.3 Inverse document frequency	5
2.4 Cosine similarity	6
2.5 Euclidean distance	6
2.6 Performance evaluation	7
2.6.1 Accuracy	7
2.6.2 Precision	8
2.6.3 Recall	8
2.7 F-measure	9
2.7.1 F-Beta measure	9
2.8 Relational databases	9
2.8.1 Database management system	10
2.9 Introduction to SQL	10
2.9.1 Queries in SQL	11
2.9.2 Select	11
2.9.3 Where	11
2.9.4 Join	12
3 Concept	13
3.1 Concept of analysing the lecture's slides	13
3.2 Concept of analysing the exercises	16
3.3 Concept of comparing slides and exercises	17
3.4 Clustering of keywords	18
4 Implementation	19
4.1 Analysing the lecture slides	19

4.1.1	Converting the slides	19
4.1.2	Analysing the slides	22
4.2	Analysing the SQL exercises	24
4.3	Comparison of slides and solutions	25
4.3.1	Baseline approach	25
4.3.2	Extending the recommendation	26
4.3.3	Restricting recommendations	26
4.3.3.1	Amount of keywords	26
4.3.3.2	Minimum cosine value	26
4.3.4	Illustration of recommendation by example	27
4.4	Challenges of the join keyword	28
4.4.1	Recognition strategy for joins	28
4.4.2	The keyword list	31
5	Evaluation	33
5.1	Evaluation setup	33
5.1.1	Keyword list	33
5.1.2	Labeling of slides and exercises	34
5.1.3	Idf variations	34
5.1.4	Performance measures	34
5.2	Baseline evaluation	35
5.3	Detecting joins	37
5.4	Clustering keywords	38
5.5	Variations in the cosine cutoff	43
5.6	Minimal cosine value	44
5.7	Discussion	45
5.7.1	Usage of keywords in the English language	46
5.7.2	Clustering keywords	47
5.7.3	Pages with many keywords	48
6	Related work	51
7	Conclusion	53
7.1	Future work	54
	Appendix	55
	Bibliography	57

List of Figures

2.1	Formula for normalized term frequency	4
2.2	Formula for normalized augmented term frequency	5
2.3	Formula for inverse document frequency	6
2.4	Formula for cosine similarity	6
2.5	Formula for euclidean distance	6
2.6	Formula for precision metric	8
2.7	Formula for recall metric	8
2.8	Formula for F-measure	9
2.9	Formula for F_β metric	9
3.1	Concept of the lecture slides analysis	15
3.2	Concept of analysing the SQL exercises	17
3.3	Concept of combining the analysis results for the lecture slides and SQL exercises together	18
4.1	Example slide 23 of the sixth chapter from the lecture in pdf format .	21
4.2	SQL exercise 2.f taken from the SQLValidator	24
4.3	Slide 11 from the second chapter which is among the best recommendations	28
4.4	Page 21 from chapter six expressing a join formulated with the WHERE keyword	29
4.5	Order of join statements formulated with the WHERE clause	30
5.1	Incorrect recommendation of page 24 from chapter nine to task E before clustering	40
5.2	Page 61 of the sixth chapter which should be chosen for recommendation	42
5.3	Slide 40 from chapter nine that contains a SQL keyword not used in a SQL context	47

- 5.4 Page 19 of the second chapter as an example for advantages that slides
with few keywords have at recommendation 49
- 5.5 Page 20 of chapter six which is not chosen for recommendation due to
too many keywords 50

List of Tables

2.1	Term frequency without normalization	4
2.2	Term frequency with normalization	5
2.3	Confusion matrix concept	7
2.4	Confusion matrix for a specific example model M	8
2.5	SQL relation of college faculty	10
2.6	SQL relation of a college course	10
2.7	Projection of the SQL relation expressing a college faculty	11
2.8	Result of a query including the SELECT and WHERE keyword	12
2.9	Result of joining tables 2.5 and 2.6	12
4.1	Recognized SQL keywords from slide 23 of the sixth chapter	22
4.2	Term frequency values for page 23 of the sixth chapter	23
4.3	Extension of table 4.2 with idf values for page 23 from the sixth chapter	23
4.4	Calculation of tf*idf values for page 23 from chapter six	24
4.5	Tf*idf analysis of example task 2.f	25
4.6	Complete tf*idf analysis for two possible recommendations in regards to task 2.f	27
4.7	Keyword recognition for page 21 from chapter six	29
4.8	Keyword recognition of page 21 from chapter six after enabling join detection	31
5.1	Selected SQL keywords for the keyword list	34
5.2	Confusion matrix of baseline approach with idf_{sub}	36
5.3	Confusion matrix of baseline approach with idf_{col}	36
5.4	Performance metrics for baseline approach	37
5.5	Results of activated join detection compared with baseline approach .	38

5.6	Performance comparison with and without join detection	38
5.7	Confusion matrix of cluster application	39
5.8	Performance comparison with and without clustering	39
5.9	Keyword analysis for task E from the SQLValidator	40
5.10	Keyword analysis of the incorrectly referred page 24 from chapter nine	41
5.11	Keyword analysis for page 61 from chapter six	42
5.12	Keyword analysis for page 24 from chapter nine after clustering . . .	42
5.13	Confusion matrix for cosine-cutoff = 0.75	43
5.14	Confusion matrix for cosine-cutoff = 0.5	43
5.15	Confusion matrix for cosine-cutoff = 0.25	43
5.16	Performance metrics for evaluated cosine-cutoff rates	44
5.17	Performance metrics for evaluated minimal cosine values	45
5.18	Tf*idf analysis for page 40 from chapter nine which features a SQL keyword in a non SQL context	46
5.19	Tf*idf analysis for page 19 from chapter two which is incorrectly chosen as recommendable for task 2.a	48
5.20	Tf*idf analysis for page 20 from chapter six which should be chosen for recommendation for task 2.a	49
A.1	Results of activated join detection compared with baseline approach for idf_{sub}	55
A.2	Comparison of metrics with and without join detection for idf_{sub} . . .	55

List of Code Listings

4.1	Process of splitting a chapter into its subchapters	20
4.2	Method inside the page class to calculate tf values	22
4.3	Method to recognize joins using the WHERE keyword	30

1. Introduction

Databases are used almost everywhere in our modern data driven world. The most used type of databases are relational databases for which there is the standardized and widely used *structured query language* (SQL) to interact with. According to a survey conducted in 2020 by Stack Overflow, a highly popular website among computer scientists and software engineers, SQL is the third most used programming language¹. As such SQL is being taught to computer science students at universities all over the world. In order to support the students at learning SQL, there have been several tools developed that provide the opportunity to solve SQL related exercises online. One of those web-based tools is the SQLValidator developed by the Research Group Databases and Software Engineering (DBSE) at the Otto von Guericke University [Obionwu et al., 2021]. The SQLValidator is integrated in the database courses held by the DBSE and encompasses exercises, questionnaires and tests to assess the students' SQL programming skills. By using the SQLValidator students and course administrators profit alike because the students receive immediate feedback on their exercises while the workload of administrators is reduced and thus can be used for more personal teaching. The DBSE group compared the exam results of students who used the SQLValidator with those of students who did not use it and came to the conclusion that students benefit from interacting with the validator. Hence the SQLValidator is subject to ongoing development with more use cases to be implemented in the future in order to provide a better learning experience for students. One opportunity for improvement is the feedback students receive when submitting a query. Currently, the students are informed whether they solved a task correctly or not. Additionally, the result of a student's query is presented as a table next to the table of the desired solution. However, students that have trouble completing a task need additional information, for instance in which SQL domain they lack knowledge.

¹ <https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-all-respondents>, last access on 29.07.2021

Goal of this bachelor thesis

This bachelor thesis aims at providing more detailed feedback for students that have problems solving the exercises. Therefore, the feedback that students receive after submitting their queries has to be enriched with information that points the students to SQL topics they have problems solving. Our feedback will additionally contain a recommendation about which slides of the lecture are necessary to successfully solve a task. The process of searching for the best slides of the lecture to describe the content of a particular exercise will be done by computing the cosine similarity between slides and exercises and choosing the best pairs. The content of this thesis encompasses the concept as to how a similarity can be derived between the lecture's slides and SQL exercises. This concept will then be implemented as a prototype that includes various optimization techniques. Finally, we analyse the performance of our prototype and derive conclusions about further improvements.

Structure of the Thesis

- **Background Chapter:**
Provides necessary information about every technology utilized in this bachelor thesis including relational databases, the cosine similarity technique, performance metrics and core SQL concepts.
- **Concept Chapter:**
Visualizes the workflow of the recommendation process. Starts with the process of making the lecture's slides and exercises comparable with each other and in the following explains the step by step process of how we derive a similarity between both.
- **Implementation Chapter:**
Explains the implemented methods of our prototype based on examples. Core concepts of the implementation are presented in depth and challenges during the implementation are talked about.
- **Evaluation Chapter:**
Evaluates the performance of our prototype based on the methods shown in the implementation chapter by beginning with a baseline approach that gets incrementally extended with several optimization techniques.

2. Background

This chapter aims to provide the reader a basic understanding of the technologies relevant to this bachelor thesis. The goal of this thesis is to develop a system for extracting key elements of SQL queries and map them to slides of the lecture which are explaining the topics needed to understand the query. In this use case, the content of queries and the lecture slides are represented by SQL keywords. These keywords are recognized and counted. A term weighting system is used to gauge the topic of queries and slides by increasing or decreasing the relevance of each encountered keyword. In the next step, the mapping of queries and slides is done based on similarity derived by calculating the cosine similarity. The mapping of queries to slides is then evaluated by using the parameters precision, accuracy, recall, F-measure and F_β -measure. During this process, a database management system is used for storing the computed data such as the occurrences of keywords in the slides and queries. In the subsequent subsections, we briefly discuss these technologies.

2.1 Tf*idf

Tf*idf (*term frequency times inverse document frequency*) is used as a term weighting system to calculate a weight for each encountered term in a document. The tf*idf value for each term is derived by multiplying the *term frequency* (tf) and the *inverse document frequency* (idf). In this thesis different methods for calculating the term frequency will be presented. Nonetheless, a feature that all these methods have in common is that the value of the term frequency for a term t is being influenced by the number of occurrences of the term t . A higher number of occurrences results in a higher tf value. Solely relying on the term frequency to extract the meaning from a document is not advisable. Suppose there is a document collection which features various newspaper articles. Due to the nature of the English language purely counting the occurrences of terms will probably result in terms such as "and", "the" or "in" receiving high tf values. Since those terms appear in the majority of documents they do not provide a meaningful way to distinguish between them. The inverse document frequency is used in order to counteract high tf values from words that do not contribute much to the meaning of the document. Additionally, the inverse

document frequency awards terms which have few occurrences within the document with a higher idf value. In our use case, we are calculating the $tf \cdot idf$ values from each slide and query. This generates a list containing the $tf \cdot idf$ values for each slide and query. Those lists can be seen as word vectors which then can be compared with each other to determine the similarity.

2.2 Term frequency

There are several ways to compute the term frequency of words. The simplest approach is to count the occurrences of terms in a document. In this context, the term frequency tf_{ij} of a term i in a document j is being defined as the number of occurrences of the term i in the document j [Manning et al., 2008]. However, research has developed more elaborate variants of the term frequency including maximum tf normalization and augmented term frequency. In the following subsections, these two methods will be briefly discussed.

2.2.1 Maximum tf normalization

A popular variant of the term frequency is the maximum tf normalization technique mentioned by Leskovec et al. [2014]. Each tf value f_{ij} is normalized by dividing it by the term frequency of the most occurring term in the document $\max_k f_{kj}$ as can be seen in Figure 2.1.

$$tf_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Figure 2.1: Formula for normalized term frequency

The reason for normalizing the tf values is that larger documents are more likely to repeat terms which results in higher tf values in comparison to smaller documents. Especially document collections with a significant deviation in the length of items are affected [Salton and Buckley, 1988]. This anomaly is explained by Leskovec et al. [2014]. Imagine a document d_1 consisting of the terms hello and world. A new document d_2 is created by inserting the content of d_1 twice into d_2 resulting in $d_2 = \{\text{hello, world, hello, world}\}$. The values of $tf_{\text{hello},d_2} = 2$ and $tf_{\text{world},d_2} = 2$ are twice as high as $tf_{\text{hello},d_1} = 1$ and $tf_{\text{world},d_1} = 1$ although the relevance of the terms did not changed. Since hello and world occurs twice in d_2 , applying the normalization results in $tf_{\text{hello},d_2} = \frac{2}{2} = 1$. In Table 2.1 the counts of the words "hello" and "world" are displayed alongside their tf values for documents d_1 and d_2 . Without the normalization the tf values for the words "hello" and "world" in d_2 are twice as high as the tf values for those words in d_1 . In Table 2.2 the tf values of d_1 and d_2 are equal since they have been normalized.

	count_{d_1}	tf_{d_1}	count_{d_2}	tf_{d_2}
hello	1	1	2	2
world	1	1	2	2

Table 2.1: Term frequency without normalization

	count_{d1}	tf_{d1}	count_{d2}	tf_{d2}
hello	1	$1/1 = 1$	2	$2/2 = 1$
world	1	$1/1 = 1$	2	$2/2 = 1$

Table 2.2: Term frequency with normalization

Salton and Buckley [1988] conducted an experiment in which they analysed six document collections varying in size and topic. For each collection there was a set of queries of different lengths. A query is supposed to be linked with certain documents from a collection. Different term-weighting methods are used for analysing both the documents and queries. The research showed that the performance of a certain term-weighting method depends on the properties of the document and the queries. For instance short query vectors perform better in combination with a term-weighting method which increases the term weights with a factor α .

2.2.2 Augmented normalized term frequency

The augmented normalized term frequency is characterized by introducing a smoothing factor α with a value in the interval of $[0 - 1]$. The usage of the smoothing factor increases the lower bound of possible tf values from $[0 - 1]$ to $[\alpha - 1]$. Thus α emphasizes terms that would generate low tf values with the regular tf formula. In other words, the mere occurrence of a term in a document is valued higher than with the tf formulas explained earlier. Manning et al. [2008] used a smoothing factor of 0.4, while other papers such as Salton and Buckley [1988] used a smoothing factor of 0.5. A lower α value shifts the focus to the actual amount of occurrences of a term in the document. As α increases the relevance of the number of occurrences decreases. With an α value of 1 each term occurring at least once would be weighted equally regardless of how many times it is present in the document. This affects the overall precision. The formula for this method is shown below in Figure 2.2.

$$\text{ntf}_{t,d} = \alpha + (1 - \alpha) \frac{\text{tf}_{t,d}}{\text{tf}_{\max}(d)^\alpha}$$

Figure 2.2: Formula for normalized augmented term frequency by Manning et al. [2008]

2.3 Inverse document frequency

A popular heuristic approach to better discriminate documents from each other based on term occurrences is the inverse document frequency. The inverse document frequency (idf) was first conceived by Karen Spärck Jones in 1972 as a way to calculate how common or uncommon a term is in a collection of documents. The mere occurrence of a term in a document gives little or no insight into the document with respect to its topic. Whereas a term that appears seldom in the collection but occurs frequently in a specific document provides a hint that the observed document

is distinguishable from other documents. The idf value for each term t_i is calculated by dividing the corpus (number of documents) N by the number of documents in which the term n_i appears and applying the logarithm. The formula for the inverse document frequency is shown below. The term frequency multiplied with the result of the formula below equals the $tf \cdot idf$ value.

$$idf(t_i) = \log \frac{N}{n_i}$$

Figure 2.3: Formula for idf [Robertson, 2004]

2.4 Cosine similarity

The cosine similarity calculates the angle between two word vectors. In our use case, the word vectors consist of keywords recognized from a query or lecture slide. The calculation of the angle is shown in Figure 2.4. The dot product of two vectors a and b is divided by the lengths of both vectors. A low angle between those vectors means that the content of the vectors is similar while a high angle expresses dissimilar content. An advantage of using the cosine angle as a similarity metric is that the length of the vectors is not relevant. In our use case, we employed this method.

$$\text{cosine}(a, b) = \frac{a \cdot b}{\|a\| \cdot \|b\|} = \frac{\sum_{i=1}^N a_i \cdot b_i}{\sqrt{\sum_{i=1}^N a_i^2} \sqrt{\sum_{i=1}^N b_i^2}}$$

Figure 2.4: Formula for cosine similarity by Sidorov et al. [2014]

2.5 Euclidean distance

Another notable method is the euclidean distance. Using this metric, the similarity between word vectors is calculated by measuring the distance between those two vectors. The distance d between the points x and y in a n -dimensional space is derived by squaring the difference between x and y for each dimension and applying the square root. The formula for this method is shown below in Figure 2.5. A drawback in employing this metric is that the length of the vectors affects the similarity result.

$$d([x_1, x_2, \dots, x_n], [y_1, y_2, \dots, y_n]) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}$$

Figure 2.5: Formula for euclidean distance by Leskovec et al. [2014]

2.6 Performance evaluation

The performance of the recommendation is measured by the evaluation metrics accuracy, precision, recall, F-measure and the F_β -measure. For this purpose, a confusion matrix is created which counts the occurrences of true positive, false positive, true negative and false negative values. For each correct prediction the counter for true positives is incremented by one. A false positive occurs if a page which should not be recommended for an exercise is being recommended to the student. True negatives are all the pages which are correctly not recommended for an exercise. False negatives are pages which are not recommended although they should be recommended. For each exercise, there is only a small number of slides that share the same topic with a particular exercise. This number compared to the total number of available slides is small. This leads to a high estimation of the number of true negative values. The confusion matrix concept is shown in Table 2.3.

	pred. pos.	pred neg.
actual pos.	TP	FN
actual neg.	FP	TN

Table 2.3: Confusion matrix concept

2.6.1 Accuracy

The accuracy parameter calculates the frequency with which a model provides a correct classification. The accuracy is calculated as shown below [Tan et al., 2005]:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

The formula for binary classification problems is:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

A disadvantage of using accuracy is that it might be misleading regarding data sets with unevenly distributed instances. To illustrate this problem, we introduce a hypothetical model M that analyses a data set consisting of 100 instances unevenly distributed among two classes cancer and \neg cancer. Among those 100 instances, ninety belong to class \neg cancer and ten belong to class cancer. Our model M is labeling each instance as a member of class \neg cancer. Only taking the accuracy into account for evaluation reveals the disadvantage of using accuracy as the sole evaluation parameter. The accuracy of model M is $\frac{90}{90 + 10} = 0.9$, indicating a well-performing model although there is not a single instance of class cancer correctly labeled. None of the sick patients are correctly diagnosed based on the classification of model M. Hence we need to use additional performance metrics such as precision or recall for evaluation. The confusion matrix for model M is shown in Table 2.4.

Model M	pred. pos.	pred neg.
actual pos.	90	0
acutal neg.	10	0

Table 2.4: Confusion matrix of model M

2.6.2 Precision

The precision parameter calculates the proportion of true positive predictions out of all positive predictions. Possible values for the precision are in the interval of $[0 - 1]$. A perfect precision value is 1 because in this case every instance predicted as true is correctly labeled and thus relevant. The worst possible prediction value is zero. This is the case if either there are no positive predictions or none of the instances labeled as true were correctly predicted. Applying the precision formula in Figure 2.6 for the hypothetical model M mentioned in Section 2.6.1, we measure a precision value of $\frac{0}{0+0} = 0$. A precision value as low as this provides a hint for the user that the models' predictions need to be further analysed.

$$Precision = \frac{TP}{TP + FP}$$

Figure 2.6: Formula for precision by [Ceri et al. \[2013\]](#)

2.6.3 Recall

The recall parameter derives the fraction of relevant instances that were retrieved from the number of instances predicted as true. Calculating the recall value of model M by using the formula presented in Figure 2.7 leads to $\frac{0}{0+10} = 0$. This concludes model M is unable to retrieve a single relevant instance, in this case, cancer patients from the data set. In [Buckland and Gey \[1994\]](#) it is stated that often recall and precision are in a trade-off relationship which causes one parameter to decrease in value as the other one increases. The relevance of precision and recall depends on the context they are used in. In the context of detecting cancer in patients it is desirable to have a high recall value to the expense of the precision. The consequences of a cancer patient not being correctly diagnosed are way more severe than a patient receiving a false positive diagnosis who has then to get examined a second time. We will discuss our observations of this correlation in the evaluation chapter.

$$Recall = \frac{TP}{TP + FN}$$

Figure 2.7: Formula for recall by [Ceri et al. \[2013\]](#)

2.7 F-measure

The F-measure is a performance metric that equates to the harmonic mean of the metrics recall and precision. By using the F-measure one combines the precision and recall into one single metric. The highest possible value for the F-measure is 1.0 and the lowest value is 0. The formula to calculate the F-measure is shown in Figure 2.8.

$$F = \frac{2 \times \text{Recall} \times \text{Precision}}{(\text{Recall} + \text{Precision})}$$

Figure 2.8: Formula for F-measure by Sammut and Webb [2011]

2.7.1 F_β -measure

A deviation of the F-measure described above is the F_β -measure that is defined as shown in Figure 2.9. The β parameter is used to influence whether a high precision or recall value is more important. With a β of 1.0 the formula equals the F-measure shown in the section above in which precision and recall are seen as equally important. A β below 1.0 puts more emphasis on precision and as β approaches zero only the precision value has an influence on the result. The opposite is the case for a β above 1.0 since then the recall value is more dominant.

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \quad (0 \leq \beta \leq +\infty)$$

Figure 2.9: Formula of F_β by Sasaki [2007]

2.8 Relational databases

Relational databases were first mentioned by Codd [1970]. As the term relational implies the underlying logic of these kinds of databases is based on the mathematical relation theory. In relational databases relations are expressed as tables. The heading of a table is a set of columns, which are called attributes in relation theory. The body of a table consists of rows with each row representing a tuple. Two tables 2.5 and 2.6 are shown below. Table 2.5 illustrates how a faculty for computer science could store data about their students. The table heading contains the columns: first_name, last_name, student_id, program and average grade. The second table 2.6 stores information about the attending students of the database course. Therefore the table heading of 2.6 contains the student_id and the students' grades.

first_name	last_name	student_id	program	average grade
John	Doe	10000	BI	3.0
Arthur	Dent	10001	CS	2.7
Bruce	Wayne	10002	CS	1.3
Uther	Lightbringer	10003	CVS	2.0
Luke	Skywalker	10004	CSE	3.3

Table 2.5: Faculty_Table

student_id	grade
10000	4.0
10001	1.3
10002	3.7
10003	2.0
10004	5.0

Table 2.6: Course_Table

2.8.1 Database management system

A database management system (DBMS) is a software which is used by the user or an application program to access a database. The DBMS offers a wide range of functionality including an optimized performance for database operation, data recovery, restricted user access and more [Saake et al., 2018]. In the context of this thesis we had to access already existing relational databases that stored data relevant for our task. For this purpose, we used MySQL as an open source DBMS in combination with the web interface phpMyAdmin that offers various functions to effectively work with SQL databases.

2.9 Introduction to SQL

The structured query language (SQL), formerly known as SEQUEL, was published by Chamberlin and Boyce [1974]. Today SQL is a standardized and widely used query language to interact with relational databases. SQL statements can be categorized into different sublanguages². There are Data Manipulation Language (DML) statements altering the stored data. For example, with the update statement the value of certain columns can be changed. With Data Definition Language (DDL) statements the user is able to define and modify data structures. DDL statements include the CREATE TABLE and CREATE VIEW statements among others. The third supported sublanguage is the Data Control Language (DCL) for controlling the access to the data. In order to query the DBMS for data, SQL offers Data Query Language statements such as the SELECT statement.

² <https://www.learnsteps.com/differentsublanguagesofsql/>, last access on 29.07.2021

2.9.1 Queries in SQL

By creating SQL queries the user is able to retrieve stored data from the database. The scheme of these queries is often abbreviated with the letters SFW with each letter indicating a specific SQL clause [Saake et al., 2018]. SFW, used in a SQL context, stands for the SQL commands SELECT, FROM and WHERE which are displayed below.

```
select [desired columns]
from [relation]
where [condition to restrict affected rows]
```

2.9.2 Select

The SELECT clause specifies which columns of a table will be retrieved in the query. The number of affected rows can hereby be restricted by utilizing the WHERE keyword in the query as we will show in section 2.9.3. In terms of relational theory the SELECT statement is a projection. The general structure of a SELECT statement is shown below.

```
select [column_names]
from [table_name]
```

Considering the table 2.5, a query to extract the names and program from each student of the Faculty_Table is created as follows.

```
select first_name, last_name
from Faculty_Table;
```

The resulting relation is displayed in Table 2.7 and contains the columns first_name, last_name and program.

first_name	last_name	program
John	Doe	BI
Arthur	Dent	CS
Bruce	Wayne	CS
Uther	Lightbringer	CVS
Luke	Skywalker	CSE

Table 2.7: Projection of the table 2.5

2.9.3 Where

The WHERE keyword enables the user to specify a condition that each retrieved tuple has to fulfill. In terms of the relational algebra the WHERE clause is used as a selection that restricts the resulting tuples of a query.

```
select first_name, last_name
from Faculty_Table
where program = 'cs';
```

The resulting relation contains the columns `first_name`, `last_name` and `program`. Each tuple of the relation shown in Table 2.8 satisfies the condition that the entry in the column "program" has to be equal to "cs".

first_name	last_name	program
Arthur	Dent	CS
Bruce	Wayne	CS

Table 2.8: Result of SELECT query with WHERE condition

2.9.4 Join

By using join statements multiple tables can be linked together [Saake et al., 2018]. There are several types of joins eg. LEFT JOIN, RIGHT JOIN, NATURAL JOIN and more which connect the tables in different ways. The NATURAL JOIN is often simply referred to as JOIN. A NATURAL JOIN combines tables based on equal values in columns which are present in the input tables. For example, the tables 2.5 and 2.6 can be linked together by using a NATURAL JOIN which utilizes the `student_id` column. The result is a table that combines each distinct columns from both tables.

```
select *
from Faculty_Table
natural join Course_Table;
```

first_name	last_name	student_id	program	average grade	grade
John	Doe	10000	BI	3.0	4.0
Arthur	Dent	10001	CS	2.7	1.3
Bruce	Wayne	10002	CS	1.3	3.7
Uther	Lightbringer	10003	CVS	2.0	2.0
Luke	Skywalker	10004	ENGCS	3.3	5.0

Table 2.9: Result of joining tables 2.5 and 2.6

Another way to formulate a join in SQL is utilizing the WHERE clause. Thus the query with the relation of Figure 2.9 can be formulated as follows:

```
select *
from Faculty_Table, Course_Table
where Faculty_Table.student_id = Course_Table.student_id;
```

3. Concept

In this chapter we present a workflow that shows how we link the SQL exercises to the respective lecture slides. This section is organized as follows. Section 3.1 explains how the relevant information in the slides is extracted and processed. Section 3.2 describes the step by step procedure of how the topic of a SQL exercise is obtained and processed. After both the slides and exercises are converted into a comparable format, we then visualize the procedure for mapping the slides and exercises with each other in Section 3.3. Finally, in Section 3.4 we explain an additional idea for improving the recommendations.

3.1 Concept of analysing the lecture's slides

In this bachelor thesis we analyse the slides of the lecture *Database Concepts* held by Prof. Gunter Saake at the *Otto von Guericke University* in Magdeburg. The lecture is directed at computer science students as an introduction to databases. Topics from the lecture include general concepts about databases, database design, relational theory and the query language SQL. We aim to support the students at solving SQL exercises through the recommendation of slides from the lecture that relate to the exercise they try to solve. Because of this we are restricting the analysis of the lecture slides to the chapters that contain SQL query related information required in the exercises. In the following, we will present the concept of analysing the lecture slides shown in Figure 3.1 by explaining each step of the process.

Slide conversion

At first, we select the relevant chapters of the lecture with each chapter being stored as a pdf file. In order to extract the useful information from the chapters we convert each chapter into a string format that enables us to further examine the textual content. In this string format, the beginning and end of respective subsections are not distinguishable. Since our aim is to examine single lecture slides, we organize the respective strings resulting from each chapter into smaller units. Hence the chapters are separated into their subchapters which are further grouped into smaller string

units. We are able to separate the chapters into smaller units by utilizing certain characteristics about the slides such as the occurrence of the subchapters' headline at the beginning of every page. At this point, we are in the transition from the first to the second step of Figure 3.1, which visualizes the procedure by using the first chapter as an example that is divided into its subchapters.

Preprocessing

In the process of splitting up the chapters, we perform preprocessing techniques such as removing irrelevant symbols including exclamation marks, multiple whitespaces or single letter words. The preprocessing is performed to simplify the later following analysis. In order to be able to recommend single slides to students, we have to further separate the subchapters into their single slides. The second subchapter of the first chapter is divided into its pages during the shift from the second to the third step of Figure 3.1.

Keyword recognition

After the subchapters are partitioned into their single pages, we start with the content analysis of the pages. Our concept of deriving the topic of both slides and SQL exercises is to detect which SQL keywords appear in them. We scan each page for the occurring SQL keywords by using a list of relevant keywords that is defined beforehand. The number of occurrences is stored for each keyword but this sole number of keyword occurrences in a page is not sufficient for deriving the topic of a slide. Suppose that a slide contains a keyword that appears frequently in the lecture such as the SELECT statement and a keyword that appears seldom like the JOIN clause. If there is no other information than the amount of occurrences of both keywords in the page, we would think that the pages belong to the topics SELECT and JOIN equally. In reality, the SELECT statement is contained in most of the pages and because of this it is not suited to distinguish a page from others based on this keyword. However, the JOIN keyword appears rarely in the slides and therefore it is possible to distinguish pages that contain the JOIN keyword from other pages that do not contain it. The keyword detection is shown in the third step of Figure 3.1. This lists the occurrences of the keywords SELECT, WHERE and JOIN for a hypothetical page 1. The SELECT keyword appeared twice in the page while the WHERE and JOIN keyword appeared once.

Keyword analysis

In the fourth step, we use the result of the previous keyword analysis to calculate the tf, idf and tf*idf values for each page. Through the computation of these values we get a better grasp of the subject of each page. The term frequency calculates the number of occurrences with which a term appears in a page. A more elaborate alternative to the raw count of keywords is the maximized term frequency in which the occurrence of a keyword is divided by the number of occurrences from the keyword that appeared most often in a page. More information about the term frequency can be found in Section 2.2. An advantage of using the maximized tf value instead of the raw count is that the length of the page becomes irrelevant. This is beneficial in our

case since the pages are not of equal length. The inverse document frequency assigns a weight to each keyword based on the number of pages the keyword appears in. A keyword like SELECT that is featured in most of the pages receives a lower weight compared to a keyword like JOIN which appears rarely. Figure 3.1 shows the idf values for the three keywords WHERE, SELECT and JOIN. The WHERE keyword appears less often than the SELECT clause and the JOIN keyword is from these three keywords the least often featured in the slides. The JOIN keyword receives a weight of 1.2 while the WHERE and SELECT clauses receive an idf value of 0.5 respectively 0.2. The tf and idf value for each keyword is then multiplied together to compute the tf*idf value. The tf*idf values resemble the influence each keyword of a page has in the later following computations. For the example shown in Figure 3.1 the JOIN keyword has the highest tf*idf value with 0.6 compared to 0.25 for the WHERE clause and 0.2 for the select keyword. Thus we estimate the topic of the page to be mostly about joins.

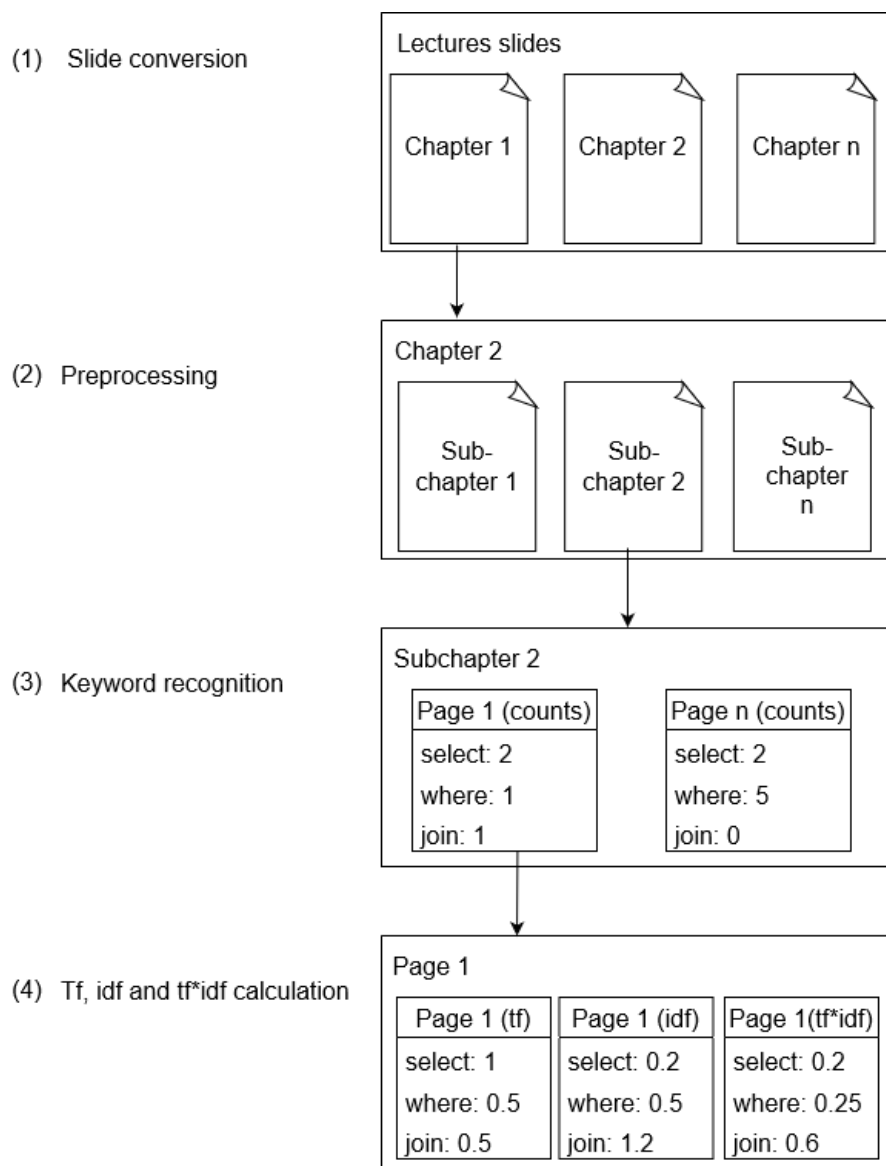


Figure 3.1: Concept of the lecture slides analysis

3.2 Concept of analysing the exercises

In order to support the students at solving these exercises it is necessary to know which parts of SQL are featured by the exercises. Our strategy for extracting the topic of an exercise is similar to the topic extraction of the lecture slides. We analyse the appearances of SQL keywords in the solutions of the exercises and by computing the tf, idf and tf*idf values, we estimate which keywords describe the exercise best. The procedure of analysing the exercises is presented in Figure 3.2.

Querying the stored solutions

The solution to each SQL exercise is stored in a database along with other information about the exercises such as an id or whether the exercise is directed at English or German speaking students. For our use case, we are only interested in the solutions and their ids for each exercise. We extract the data by querying the database and then store the extracted data in an associative array with the key being the id of an exercise and the value of this index containing the solution as a string. There is just a minor preprocessing step involved in which whitespaces are added to separate some words of the solution from each other. In comparison to the lecture slides there is no need for further preprocessing since the solutions are already formatted. This is conceptualized in Figure 3.2 between the first and second step.

Keyword recognition

After the solutions are stored in the array we have to scan each solution for the occurrences of SQL keywords by using a list of relevant SQL keywords. In the third step of Figure 3.2 we use the second exercise as an example. The exercise contains three keywords SELECT, WHERE and FROM, each appearing once.

Keyword analysis

In the next step, we calculate the tf, idf and tf*idf values for each keyword of a page. Our prototype further offers the option to cluster several keywords together. On defining a cluster we scan the keywords inside it to determine which of the respective keywords has the highest tf*idf value and we assign this value to every member of the cluster. Considering the exercise displayed in Figure 3.2, the keyword that has the highest idf weight is the WHERE keyword with the SELECT and FROM keywords having a lower idf value. The tf*idf computation shows that the topic of the page is about the usage of the WHERE keyword.

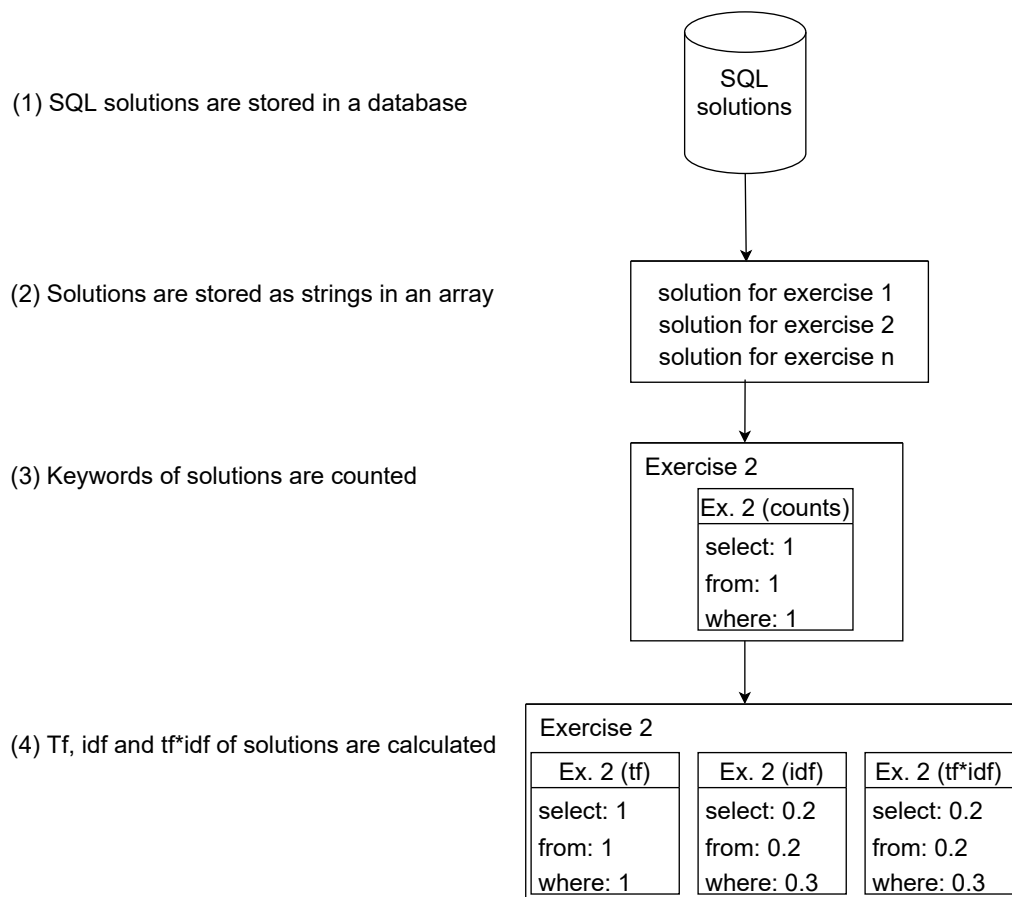


Figure 3.2: Concept of analysing the SQL exercises

3.3 Concept of comparing slides and exercises

In the previous sections 3.2 and 3.1 we described the process of converting both slides and SQL exercises into a format that allows us to compare them. The comparison is done by computing the cosine similarity between the lecture slides and SQL exercises. Figure 3.3 visualizes this process of deriving a similarity by choosing the best matching page for a hypothetical exercise 1.

Merging of previous analysis

The results of the slides and exercises analysis are merged in a list during the first step of the comparison process. For each exercise, we now have access to the tf*idf values from every page with respect to this exercise. Figure 3.3 shows the mapping between exercise 1 and each page alongside the result of their keyword analysis. The three keywords SELECT, WHERE and FROM were identified in exercise 1 with their tf*idf values of 0.2, 0.2 and 0.3. Page 1 contains the same keywords and tf*idf values as exercise 1 plus the keyword > with a tf*idf value of 0.3. The second page features the SELECT and FROM keywords with tf*idf values of 0.2 and 0.5. The last page contains the ALTER and TABLE keywords with values of 1 and 0.5.

Computation of cosine similarity

In this step, the cosine similarity for each page of the lecture is calculated in regards to exercise 1. The result is that page 1 has the highest cosine similarity with 0.8 following page 2 with 0.631 and page n with 0. The page with the best cosine value, in this case page 1, is then selected to be mapped to exercise 1 and hence be recommended to students having problem with exercise 1.

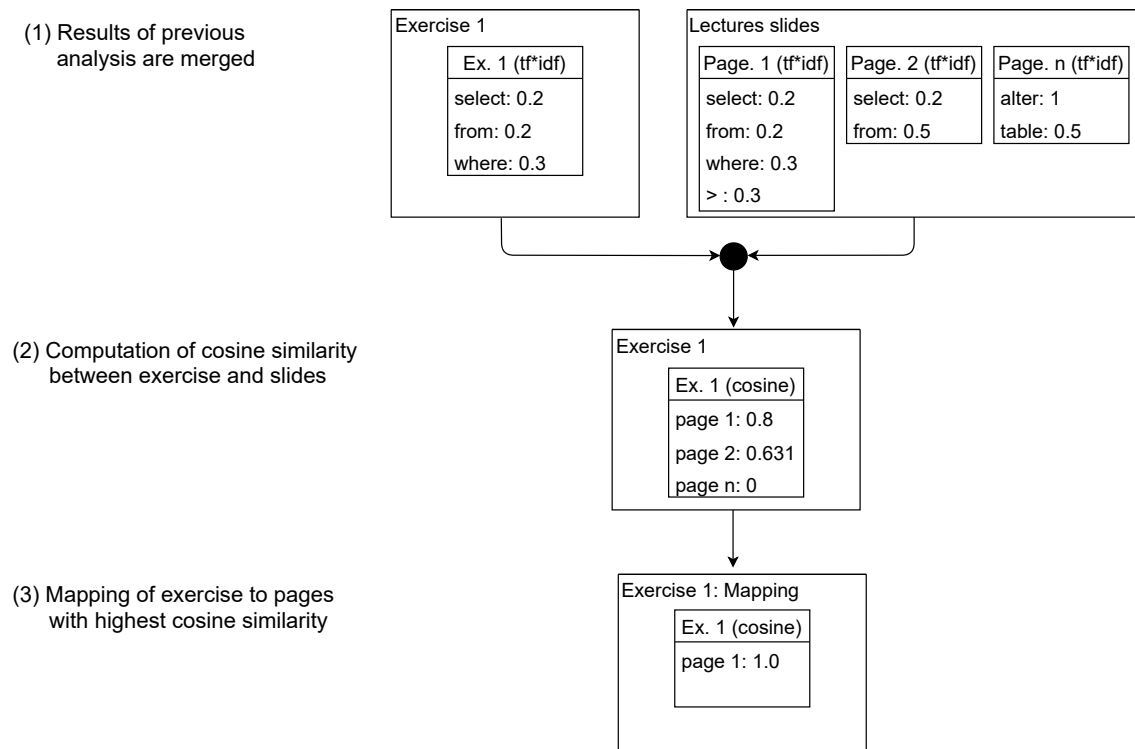


Figure 3.3: Concept of combining the analysis results for the lecture slides and SQL exercises together

3.4 Clustering of keywords

As another parameter, our prototype offers the option to create clusters consisting of an arbitrary number of keywords. After the $tf*idf$ values for each keyword are calculated the members of the cluster are scanned for the highest $tf*idf$ value. Each member of the cluster then receives the highest $tf*idf$ value. The reasoning for clustering keywords is the assumption that the occurrence of specific keywords together leads to a certain SQL topic. For instance, the keywords $<$ and $>$ are used as a range selection. Clustering both keywords together increases the $tf*idf$ value of the keyword with the lower score while other keywords of the page that are not part of the cluster remain unchanged in their $tf*idf$ values. Thus, the keywords of the cluster have a higher influence in the later following cosine calculation and it is more likely that the recommendation points us to a page about range selection.

4. Implementation

This chapter presents the implementations of this bachelor thesis. In order to better visualize the process of linking certain slides of the lecture to the solutions of SQL exercises, we provide an example for each step.

4.1 Analysing the lecture slides

4.1.1 Converting the slides

The slides of the lecture are saved in the pdf format with a file for each chapter of the lecture. Since the textual content of the slides is of interest to us we have to convert them into a string which enables us to further analyse the content. We have found that the Smalot PDF Parser Library³ suits our use case. Especially the ongoing support and development of the project is beneficial to us. After parsing the relevant files, each chapter of the lecture is represented as a string containing plain text. There are no markings whether a certain part of the text is the heading or the body of a page. In the next steps the string is partitioned into smaller strings containing coherent units such as subchapters and later on pages. For splitting the strings we are using the built-in php function `preg_split`⁴:

```
preg_split(string $pattern,string $subject,int $limit=-1):array|false
```

If a certain pattern, formulated as a regular expression, is part of a string then `preg_split` will divide the string at the occurrence of the pattern. This results in an array that saves the part before the pattern in index `n` and the part after the pattern in index `n+1`. We are utilizing this behaviour for further partitioning the content of a chapter. The process of partitioning a chapter into its subchapters is conceptually shown in Listing 4.1. Suppose the `$chapter_plain_text` variable of

³ <https://github.com/smalot/pdfparser>, last access on 29.07.2021

⁴ <https://www.php.net/manual/de/function.preg-split.php>, last access on 29.07.2021

line 2 contains the textual content of three subchapters that shall be separated from each other. The headlines of these subchapters are stored in the *\$headline* variable in line 1. Each chapter has a title page and a few introductory slides that do not belong to any subchapter. Those pages are not relevant for our analysis and thus should not be evaluated. In the first iteration of the for loop in line 5 we are separating the beginning of the chapter from the relevant content with the subchapters. This is done by applying the preg split method with the heading of the first subchapter as the *\$pattern* variable and the *\$chapter_plain_text* as the *\$subject* variable. We receive an array with the chapter introduction saved in index *n* and the remaining part of the chapter saved in the *\$remaining_text* variable. During the second iteration, we separate the first subchapter from the remaining subchapters by splitting the *\$remaining_text* at the first occurrence of the second subchapters headline. This step is performed by the code example in lines 10 to 13. After the split, the first subchapter is isolated and stored in the *\$chapter_ordered* variable of line 12. We will repeat this process for the second subchapter. After the fourth iteration *\$chapter_ordered* stores the separated subchapters in one index each.

```

1 $headlines= ['View Concept','Views in SQL','Updates via Views'];
2 // $chapter_plain_text variable contains textual content of the
  chapter;
3 $remaining_text = $chapter_plain_text;
4 $chapter_ordered = [];
5 for($i = 0; $i <= sizeof($headlines); $i++) {
6     if ($i == 0) {
7         $split = preg_split('/'. $headlines[$i]. '/',
8                             $remaining_text,2);
9         $remaining_text = $split[1];
10    }
11    elseif ($i > 0 && $i < sizeof($headlines)) {
12        $split = preg_split('/'. $headlines[$i]. '/',
13                            $remaining_text,2);
14        $chapter_ordered[$headlines[$i-1]] = $split[0];
15        $remaining_text = $split[1];
16    }
17    else ($i == sizeof($headlines)) {
18        $split = preg_split('/'. $headlines[$i-1]. '/',
19                            $remaining_text,2);
20        $chapter_ordered[$headlines[$i-1]] = $split[1];
21    }
22 }

```

Listing 4.1: Process of splitting a chapter into its subchapters

In the next step, the subchapters are divided into their pages with a page containing the content of a single slide as plain text. This is done by utilizing the fact that every page of the lecture follows the same structure with the chapter's name appearing first in the slide and the subchapter's name following. At the end of each slide there is the number of the chapter and the page number. The single pages are derived by splitting the subchapters string each time its name appears. After the completion of

these steps, there is a multidimensional array which links a chapter to its subchapters and the pages to their subchapter.

In the next step we are scanning the pages which are storing the plain text of the slides for SQL keywords by using a list of relevant SQL keywords. The plain text is hereby scanned using bi-grams. That means we use a window with a width of two words and let it slide over the textual content of a page. The usage of bi-grams lets us easier recognize SQL keywords that contain two words such as SELECT DISTINCT or NATURAL JOIN. Since we are only interested in SQL keywords, we are dismissing the plain text and store an associative array which counts the occurrences of each keyword instead.

In the following we illustrate this previously mentioned conversion of a slide in the pdf format to an array of keywords by taking the 23rd slide of the sixth chapter "The Relational Query Language SQL" as an example. The 23rd slide is part of the subchapter "The SFW Block in Detail". The core topic of this slide is the usage of the SQL LIKE operator with which the user can search for data following a certain pattern in a column. Figure 4.1 shows how slide 23 is represented in the original pdf format while table 4.1 displays the representation of the same slide solely by SQL keywords. The first column of the chapter lists the recognized SQL keywords AND, OR, LIKE and IN. The second column stores the amount of occurrences for each keyword with the IN keywords appearing twice and the other keywords appearing once.

The Relational Query Language SQL The SFW Block in Detail

Imprecise Selection

- Notation
 - `attribute like special-constant`
- Pattern matching in strings (search for multiple substrings)
- Special constant can contain the wildcard characters '%' and '_'
 - ▶ '%' stands for no character or an arbitrary string of characters
 - ▶ '_' stands for exactly one character

Saake Database Concepts Last Edited: June 11, 2020 6-23

Figure 4.1: Example slide 23 of the sixth chapter from the lecture in pdf format

Chapter: 6, Page: 23	
keywords	count
and	1
or	1
like	1
in	2

Table 4.1: Recognized SQL keywords from slide 23 of the sixth chapter

4.1.2 Analysing the slides

In the previous chapter, the textual representation of the slides has been compressed into arrays that store SQL keywords. Additionally to the occurrence of keywords in a slide we are interested in calculating the term frequency, the inverse document frequency and the $tf*idf$ values for each keyword of a slide. Since extending the multidimensional array to store the additional values is burdensome and prone to errors due to its complexity we have converted this array into objects. The array of a slides' keywords are now stored as a property of a page object, the pages of a subchapter are referenced in subchapter objects and the subchapters of a chapter are referenced in chapter objects. Ultimately the chapter objects of different chapters are referenced by a collection object. The calculation of tf values is done in the *max_tf_normalization()* method of page objects.

```

1 public function max_tf_normalization() {
2     $tf_max = 0;
3     foreach($this->keywords as $key => $value):
4         if ($value > $tf_max) $tf_max = $value;
5     endforeach;
6     foreach($this->keywords as $key => $value):
7         if($tf_max > 0) $this->tf_vals[$key] = round($value /
8             $tf_max,2);
9         else $this->tf_vals[$key] = 0;
10    endforeach;
11 }
```

Listing 4.2: Method inside the page class to calculate tf values

The function in Listing 4.2 iterates over the keyword array which stores the occurrences of keywords of a slide to find the keyword which occurred most often and saves the amount in the tf_max variable in line 4. During the next `foreach` statement in line 6 each tf value is getting normalized through the division by tf_max . Considering the example of the 23rd page of the sixth chapter, this will result in the tf values shown in Table 4.2. Term frequency values of zero are not shown.

Chapter: 6, Page: 23		
keywords	count	tf
and	1	0.5
or	1	0.5
like	1	0.5
in	2	1

Table 4.2: Term frequency values for page 23 of the sixth chapter

In the next step the idf values of those keywords need to be calculated. Our prototype offers two ways of defining the relevant corpus for the calculation of idf values for a page. The corpus can either be restricted to the subchapter to which a page belongs to or encompass the whole collection. Table 4.3 extends the analysis of the example slide by two additional columns. The fourth column of the table contains the idf values for a subchapter-wise calculation denoted as idf_{sub} while the fifth column idf_{col} stores the idf values for a collection-wise computation. The idf values for the AND and OR keywords are relatively close to each other. More interesting is the difference with the LIKE keyword that receives an idf_{sub} value of 1.041 compared to the idf_{col} value of 1.653. This difference is explained by the LIKE keyword appearing relatively often in the subchapter of page 23 and seldom in the overall collection. However, in our research, we have noticed that the collection-wise idf calculation yields better results as we will show in the evaluation chapter.

Chapter: 6, Page: 23				
keywords	count	tf	idf_{sub}	idf_{col}
and	1	0.5	0.564	0.449
or	1	0.5	0.643	0.857
like	1	0.5	1.041	1.653
in	2	1	0	0.273

Table 4.3: Extension of table 4.2 with idf values for page 23 from the sixth chapter

With the tf and idf values being computed we can calculate the $tf \cdot idf$ values for each keyword of a page. A $tf \cdot idf$ value is derived by multiplying the tf and idf value of a keyword. Table 4.4 adds the columns $tf \cdot idf_{sub}$ and $tf \cdot idf_{col}$ to the table 4.3 for the analysis of the example page. The LIKE keyword reaches the highest $tf \cdot idf$ value for both $tf \cdot idf$ columns with a value of 0.521 in $tf \cdot idf_{sub}$ and 0.827 in $tf \cdot idf_{col}$. This means that the LIKE keyword will have the most influence in the later following cosine similarity calculation.

Chapter: 6, Page: 23						
keywords	count	tf	idf _{sub}	tf*idf _{sub}	idf _{col}	tf*idf _{col}
and	1	0.5	0.564	0.282	0.449	0.225
or	1	0.5	0.643	0.322	0.857	0.429
like	1	0.5	1.041	0.521	1.653	0.827
in	2	1	0	0	0.273	0.273

Table 4.4: Calculation of tf*idf values for page 23 from chapter six

4.2 Analysing the SQL exercises

We have explained the implementation of the lecture slides analysis in the previous Section 4.1.2. Before we are able to compute a similarity between the slides of the lecture and SQL exercises we have to analyse the latter. We will use an example exercise in order to better illustrate the process of automatically extracting the domain of SQL keywords a student should know for successfully solving an exercise. Figure 4.2 displays an excerpt from the SQL exercise sheet 8 task 2.f directed at students. The task is to extract the names of employees from the table with the condition that only names are extracted that contain exactly two Ls.

Task (f) Employees with two Ls

Task Description:

Determine the names of employees that have the L twice in their name.

Hints				
Employee				
ID	name	adresse	job	salary
212	Jane Schmidt	Hamm	Steward/-esse	2650.00
312	Walter Schmidt	Hamm	Steward/-esse	2450.00
313	Gustav Schürmann	Hamm	Pilot	6450.00
314	Frank Brallier	Grevenbroich	Pilot	8450.00

Figure 4.2: SQL exercise 2.f taken from the SQLValidator

Relevant information for each exercise, including a solution to this exercise, is already stored in a relational database. These stored solutions are used by us to determine relevant SQL keywords to solve a particular exercise. We are querying the database for the SQL solutions and save them as strings in an associative array. For example, the solution for solving the task shown in Figure 4.2 is as follows:

```
Select Name FROM employee where Name LIKE '%L%L%L';
```

Since we are only interested in the SQL keywords contained in the solutions strings we are using a list of relevant keywords in combination with pattern matching to recognize these keywords. The built-in php function `preg_match`⁵ as shown below provides this utility. As a result for a keyword search inside the solution string we either receive the amount of occurrences or false if there is no occurrence.

```
preg_match(string $pattern ,string $subject) :int|false
```

Considering the example task we are recognizing the keywords SELECT, WHERE, LIKE and FROM once. We are further calculating the tf, idf and tf*idf values in order to establish a ranking between keywords with respect to their ability to describe the core topic of the exercise. Hereby the idf corpus is defined over all SQL exercises currently accessible for the students. For instance, the SELECT and the FROM keyword are not suited to describe this particular exercise because most of the solutions to exercises include both keywords. Instead, the LIKE keyword should receive a higher weight because it is used less often and therefore we are able to distinguish this exercise from the other exercises. In Table 4.5 the computed values for each recognized keyword are shown. The tf*idf value hereby resembles the influence a keyword has on the later following cosine comparison. In the chosen example it shows that the LIKE keyword will influence the comparison decision significantly.

Example exercise 2.f				
keywords	count	tf	idf	tf*idf
select	1	1	0.087	0.087
where	1	1	0.301	0.301
like	1	1	1.041	1.041

Table 4.5: Tf*idf analysis of example task 2.f

4.3 Comparison of slides and solutions

A recommendation of a particular slide for a SQL exercise is based on the cosine similarity between those two. In the following section, we introduce our baseline approach for recommending pages to the students. We have extended this baseline approach to improve the performance of our recommendations. These extensions are explained in the Sections 4.3.3.1 and 4.3.3.2.

4.3.1 Baseline approach

The baseline approach encompasses the process of finding the best matching page for each exercise. In order to find the best match for an exercise, we have to iterate through every page and calculate the cosine similarity with respect to the exercise. The best match is defined by the page that reaches the highest cosine value.

⁵ <https://www.php.net/manual/de/function.preg-match.php>, last access on 29.07.2021

4.3.2 Extending the recommendation

In some cases only recommending the best match might not be sufficient because there could be several viable pages to recommend. Therefore we have implemented a parameter in our prototype that enables the user to set a threshold below the highest cosine similarity. Pages that are above this threshold are then also recommended. Suppose there is an exercise A for which we compute the cosine similarity among all pages and the three pages with the highest cosine values are $\text{cosine}(\text{page 1}) = 0.9$, $\text{cosine}(\text{page 2}) = 0.81$, $\text{cosine}(\text{page 3}) = 0.5$. The threshold is defined as a fixed percentage of the best matching cosine value. If the threshold is $t = 0.6$ only cosine values that reach at least sixty percent of the highest cosine value are being recommended. For the three pages this results in the recommendation of page 1 since it has the highest cosine value and page 2 because the value is above the threshold t with $\frac{0.8}{0.9} = 0.9 > 0.6$.

4.3.3 Restricting recommendations

In the prototype's current state there will always be at least one recommendation for each exercise. This behaviour is not desirable and in the sections below we explain under which circumstances the prototype should be restricted from recommending pages.

4.3.3.1 Amount of keywords

Our search for SQL keywords is applied to every page of the lecture. Due to the nature of the lecture's slides, there are pages that contain certain words that are by coincidence also SQL keywords such as IN, AND or AS. We have encountered in our research that pages with a single keyword that is not used in a context of SQL mislead our recommendation. Therefore, we implemented an option to only recommend pages that contain more than a certain amount of keywords.

4.3.3.2 Minimum cosine value

Some SQL exercises for the students feature SQL concepts which are not part of the lecture slides. In these cases, the calculation of the cosine similarity often yields comparably low cosine values. Suppose there is an exercise for which there is not a single page having a cosine value above zero. In this case, the prototype would recommend every page to the user since the highest cosine value of zero is shared by all pages regarding this exercise. A less severe case would be an exercise for which the highest cosine value is comparably low, for example, 0.1. The prototype would then recommend this page although the similarity between page and exercise is rather low. Instead of recommending a page that is not fitting for the exercise, we chose to not recommend any page. For this purpose, we implemented an option to set a minimal cosine value for pages that has to be reached in order for those pages to be recommended.

Chapter: 6, Page: 23, Cosine: 0.74				
keywords	count	tf	idf	tf*idf
and	1	0.5	0.564	0.282
or	1	0.5	0.643	0.322
like	1	0.5	1.041	0.521
in	2	1	0	0

(a) Tf*idf evaluation for the desired slide 23 from chapter six

Chapter: 2, Page: 11, Cosine: 0.407				
keywords	count	tf	idf	tf*idf
and	4	1	0.368	0.368
or	1	0.25	0.544	0.136
like	1	0.25	0.845	0.211
in	1	0.25	0.243	0.061
as	1	0.25	0.067	0.017
=	1	0.25	0.845	0.211

(b) Tf*idf values for the unwanted page 11 from the second chapter

Table 4.6: Complete tf*idf analysis for two possible recommendations in regards to task 2.f

4.3.4 Illustration of recommendation by example

A recommendation has to be made for the example exercise shown in Figure 4.2. The amount of recommended pages changes depending on which parameter settings are used in our prototype. Suppose we are using a moderate threshold $t = 0.5$ considering pages for recommendation if their cosine value exceeds $t = 0.5$. In this case, there are three pages selected for recommendation, including page 23 of the sixth chapter as the best match. Table 4.6 displays the analysis of the best and third best match considering exercise 8.2.f. The upper tabular of table 4.6 shows the analysis of the best recommendation which reaches a cosine similarity of 0.74 while the lower tabular displays the analysis for the third best recommendation which reaches a cosine similarity of 0.407. Keeping in mind Table 4.5, there are three keywords contained in the solution of exercise task 2.f: SELECT, WHERE and LIKE. The two pages described in the table below share the LIKE keyword with the solution. Thus, the similarity between a page and the solution is only influenced by the LIKE keyword and since page 23 has the higher tf*idf value for this keyword it reaches a higher cosine similarity. Compared to the best match the like keyword in page 11 influences the tf*idf value less than the like keyword in page 23.

Relational Databases – Data as Tables		SQL Data Definition	
<h2>Possible Domains in SQL</h2> <ul style="list-style-type: none"> • integer (also: integer4, int), • smallint (also: integer2), • float(p) (also, for short, float), • decimal(p,q) and numeric(p,q) with q decimal places, • character(n) (also, for short, char(n), with $n = 1$ just char) for character strings of fixed length n, • character varying(n) (also, for short, varchar(n) for variable-length character strings up to the maximum length n, • bit(n) or bit varying(n) like varchar but for bit strings, and • date, time, timestamp for specifying dates, times and the combination of date and time 			
Saake		Database Concepts	
		Last Edited: May 2, 2020 2–11	

Figure 4.3: Slide 11 from the second chapter which is among the best recommendations

The slide of page 11 from the second chapter is shown in Figure 4.3. This slide is part of the subchapter "SQL Data Definition" which explains the different SQL data types. The usage of the word LIKE is hereby not in the context of the SQL keyword LIKE. Therefore, a recommendation to the students is not desired.

4.4 Challenges of the join keyword

The recognition of SQL keywords is done by applying the `preg_match` method while using a keyword list that stores all relevant SQL keywords. This process is explained in the Sections 4.1 and 4.2. The recognition of joins however needs a more sophisticated effort because joins in SQL can either be created with the JOIN keyword or by the usage of the WHERE keyword. In case of a join in the form of the WHERE keyword, simply counting the amount of keywords would lead to an increase of recognized = and WHERE keywords but a JOIN would not be recognized. For a more thorough explanation of the SQL JOIN keyword see Section 2.9.4. We have implemented a way to recognize those disguised joins in our prototype. In the following section we will describe our process of detecting joins.

4.4.1 Recognition strategy for joins

Figure 4.4 displays a join formulated with the WHERE keyword instead of using the JOIN keyword. The usual approach of counting the keywords leads to Table 4.7.

Instead of three join keywords there are only two joins recognized. The WHERE keyword should not be encountered and the = should only appear once.

The Relational Query Language SQL
The SFW Block in Detail

Join Condition

- *Join condition* has the form:

`relation1.attribute = relation2.attribute`
- Example:

```
select Name, Vintage, PRODUCER.Vineyard
from WINES, PRODUCER
where WINES.Vineyard = PRODUCER.Vineyard
```

Saake
Database Concepts
Last Edited: June 11, 2020
6–21

Figure 4.4: Page 21 from chapter six expressing a join formulated with the WHERE keyword

Chapter: 6, Page: 21	
keywords	count
join	2
select	1
from	1
where	1
in	1
=	2

Table 4.7: Keyword recognition for page 21 from chapter six

In order to recognize the join from the example shown in 4.4 our prototype utilizes that the structure of those join statements is following an order displayed in Figure 4.5. The bold marked steps 1, 3, 5 and 7 are used in our recognition method. Since each of those joins starts with the WHERE statement our function notices the occurrence of the WHERE statement. After encountering a WHERE keyword the function is searching for the appearance of a dot symbol followed by an equal sign which in turn is followed by another dot symbol.

1. **Where keyword**
2. name of table A
3. **dot symbol**
4. name of a column
5. **equal sign**
6. name of table B
7. **dot symbol**
8. name of a column

Figure 4.5: Order of join statements formulated with the WHERE clause

Listing 4.3 shows the conceptual implementation of the join recognition. We want to detect if the keywords of a join appear in the right order. We are using the array `$join_stack` that stores the elements 'where', '.', '=', '.' in this exact order. In line 3 the content of a page is read in as one-liners. For each word we test if this word is equal to the first element in the `$join_stack`. If the word is equal to the first element we remove it from the stack. This will be repeated until the stack is empty indicating that each keyword necessary to describe a join has been found and therefore we increment the `$join_counter` variable. The `$join_stack` is being filled again with keywords and we resume the join detection until every word of the page has been read in. At the end we return the amount of detected joins stored in the `$join_counter` variable of line 12.

```

1  $join_stack = [];
2  array_push($join_stack, 'where', '\.', '=', '\. ');
3  foreach ($uni_gram as $index => $entry):
4      if(preg_match($join_stack[0], $entry) != false)
5          if(sizeof($join_stack) > 1) array_shift($join_stack);
6          else {
7              array_shift($join_stack);
8              array_push($join_stack, 'where', '\.', '=', '\. ');
9              $join_counter++;
10         }
11 endforeach;
12 return $join_counter;

```

Listing 4.3: Method to recognize joins using the WHERE keyword

Considering the example of Figure 4.4 we are now able to correctly declare the recognized keywords in Table 4.8. There are three JOIN keywords contained in the page while the WHERE keywords disappeared and there is now one = keyword being recognized.

Chapter: 6, Page: 21	
keywords	count
join	3
select	1
from	1
in	1
=	1

Table 4.8: Keyword recognition of page 21 from chapter six after enabling join detection

4.4.2 The keyword list

A keyword list is used every time we search for the appearances of certain SQL keywords. For the purpose of the evaluation we wanted to test the recommendation performance with different keyword lists. Therefore, we have implemented a tool to create customary keyword lists that contain specific SQL keywords selected by the user. Those keywords are part of a domain of possible SQL keywords. For instance, there are SQL keywords available in our prototype such as CONCAT that are currently not mentioned in the lecture. Since this could also change in the future, our prototype already supports a wide variety of keywords. Another use case for customary keyword lists are keywords that contain more than one word. Considering the SELECT DISTINCT keyword we enable the user to freely decide whether they want to recognize the SELECT and DISTINCT keywords independently from each other or whether they want to recognize the SELECT and SELECT DISTINCT keywords.

5. Evaluation

In this chapter, we will present the evaluation of our recommendation system. We will start by explaining in Section 5.1.1 how we derive a particular list of relevant SQL keywords that we will use in later sections for evaluating the performance of our prototype. Section 5.2 evaluates the performance of our prototype using a list of relevant SQL keywords. We have enriched our prototype with additional methods in order to increase the performance. One of these methods is the ability of our prototype to detect joins that are formulated with the WHERE clause. The impact that the join detection has on the performance is explained in Section 5.3. Another extension of our prototype is the clustering of keywords. The clustering extension will be evaluated in Section 5.4. In Section 5.5 we will extend our prototype with an optional setting that lowers the requirements a page has to fulfill in order to be recommended to the students. In Section 5.6 we will present the reader the effect of setting a minimal cosine value that a page has to reach in order to be chosen for recommendation. Finally, section 5.7 summarizes the performance evaluation of our prototype and shows challenges that have occurred during our research.

5.1 Evaluation setup

5.1.1 Keyword list

Essential to the analysis of the lecture slides and SQL exercises is the detection of SQL keywords. The keywords detection works by scanning the content of a slide or SQL exercise and check for each encountered word if it is contained in the list of relevant keywords. Our prototype enables the user to create these lists of relevant keywords by choosing them from a pool of SQL keywords. The keyword list has a significant influence on the mapping between slides and exercises. Therefore we propose a list of SQL keywords that will be used in the further evaluation in order to make the different implementation methods comparable. The pool of SQL keywords contains 58 elements hence the number of keyword lists that can be created by combining the elements of the pool, is estimated very high. We are not able to exhaustively test

every combination of keywords and as a result, we chose the keyword list displayed in Table 5.1 since it achieves the highest encountered performance.

Relevant SQL keywords					
select	distinct	where	and	or	not
null	update	delete	min	max	count
avg	sum	like	in	between	as
join	union	group	having	exists	any
all	case	create	<	<=	>
>=	round	=	drop	alter	constraint
unique	primary	foreign	check	default	view
concat	substring	select distinct	natural join	left join	right join
full join	primary key	foreign key	create view	create table	group by
order by	insert into	insert	order		

Table 5.1: Selected SQL keywords for the keyword list

5.1.2 Labeling of slides and exercises

In order to evaluate whether a mapping between slides and exercises is useful, we need to establish a ground truth. Thus we manually labeled the SQL exercises so that for each exercise there is a stored selection of recommendable slides from the lecture. The decision of which slides shall be recommended for a particular exercise might be topic of discussion. Depending on the experts' view, the selection of suitable slides can be either flexible or strictly. As a result, the evaluation is greatly influenced by the experts' labeling. We mitigate this problem by using multiple experts in the labeling process instead of a single expert.

5.1.3 Idf variations

We calculate an idf value for each encountered SQL keyword in the lecture slides and SQL exercises. The idf value tries to emphasize a keyword's ability to describe the content of the page or exercise it occurred in. A thorough explanation of the inverse document frequency can be found in Section 2.3. We implemented two slightly different ways of calculating the idf corpus. The corpus can be defined subchapter-wise, meaning that the number of the keyword's occurrences in the subchapter of a page matters. Alternatively, our prototype offers the option to calculate the idf value of a keyword by taking the number of occurrences for all available pages into account. In the following sections, we will refer to the subchapter-wise idf calculation as idf_{sub} and the collection-wise idf values will be referred to as idf_{col} .

5.1.4 Performance measures

We will evaluate the performance of our prototype by computing several performance metrics. In the following, we will shortly present each of the performance measures and explain how they help us assess the performance of the prototype.

Accuracy

The accuracy metric describes the fraction of correct predictions the model made by dividing the number of true negative and true positive instances by the total amount of instances. In our case, the accuracy resembles the fraction of slides that were correctly predicted as being recommendable plus the amount of slides that were correctly not recommended to the students. An accuracy of zero means that there were no correct predictions while an accuracy of 1.0 implies that there is no false prediction.

Precision

The precision metric is derived by dividing the number of true positive predictions by the amount of true positives and false positive predictions. In our context, we use the precision to estimate how likely it is that a slide recommended to students is actually meant to be recommended.

Recall

The precision metric is derived by dividing the number of true positive predictions by the amount of true positives and false negative predictions. By using the recall we receive the fraction of relevant instances that were retrieved. In our use case, the recall value tells us how many slides that are labeled as positive were actually recommended to the students.

F-measure

In the previous two sections we mentioned the precision and recall performance metrics. By using the F-measure we combine both metrics into one formula and are thus able to find a balance between them. Considering our use case we think that the precision is more important than the recall metric. A high precision value means that almost every slide that is recommended was labeled by us as useful and thus the probability of students wasting their time, studying a slide which will not improve their understanding of the exercise, is low. The F_β enables us to put more emphasis on one of the metrics and therefore we chose to weigh the precision value in a way that resembles its significance. A β value of 1.0 is identical to the f-measure mentioned above while a β value of zero only considers the precision. Therefore we chose 0.5 for our β parameter.

5.2 Baseline evaluation

The baseline evaluation of our prototype derives a mapping between slides and SQL exercises by purely computing the cosine similarity without using any additional parameters. Only the preferred way of calculating the idf values has to be selected. Using the baseline approach means that only the slide with the highest cosine similarity will be selected for recommendation. If there are multiple slides sharing the best cosine value then all of those are chosen for recommendation. The confusion matrix of the baseline approach using the idf_{sub} computation is shown in Table 5.2.

In our current implementation we have 180 slides and 66 SQL exercises resulting in 11.880 entries in the confusion matrix. Out of the 70 entries predicted as positive 38 were actually positive. There are 11.810 entries in the column predicted negative with 178 counted as false negative and 11632 as actually negative. The table cell of true negative entries is of interest to us since it contains a little more than 98% of all entries. This imbalance of instance distribution is expected because there are 180 possible recommendations for each exercise but usually only a few slides for each exercise are labeled as recommendable. Suppose there is an exercise for which we selected three pages as appropriate. If our prototype would recommend a random slide for this exercise which turns out to be inappropriate for this exercise, there would still be 176 slides correctly classified as not recommendable and because of this 176 entries are added to the true negative cell. This effect is reinforced by using the baseline approach since the recommendation is restricted to only recommend the pages with the highest cosine value, further decreasing the amount of slides that are recommended. The result for the baseline approach in combination with the idf_{col} calculation is shown in Table 5.3. The variation in the idf calculation is barely showing in the classification since the idf_{col} method predicts 66 instances as positive compared to 70 positive predictions in the idf_{sub} computation. The difference in negative predictions is also negligible with 11.810 negative predictions in Table 5.2 and 11.814 in Table 5.3.

	pred. pos.	pred neg.
actual pos.	38 (TP)	178 (FN)
acutal neg.	32 (FP)	11632 (TN)
total	70	11.810

Table 5.2: Confusion matrix of baseline approach with idf_{sub}

	pred. pos.	pred neg.
actual pos.	38 (TP)	178 (FN)
acutal neg.	28 (FP)	11636 (TN)
toal	66	11.814

Table 5.3: Confusion matrix of baseline approach with idf_{col}

Table 5.4 shows the performance metrics with respect to the confusion matrices from Table 5.2 and 5.3. The accuracy of both idf calculations is rather high with 0.982 using idf_{sub} and 0.983 using idf_{col} . This is mostly due to the previously described fact that most of the pages are correctly classified as true negative. The precision value of idf_{sub} is slightly lower than the precision for idf_{col} method with 0.576. That means slightly more than half of our prototypes recommendations are correct recommendations. Each of the remaining metrics recall, F-measure and $F_{\beta=0.5}$ are rather similar for both idf computations. The recall value for both methods is 0.176 which implies that around 17% of the slides classified as recommendable are actually selected by the prototype. The F-measure, which is influenced by the precision and recall metrics

equally, reaches 0.266 with the idf_{sub} and 0.27 with the idf_{col} . The most important metric in our use case is the $F_{\beta=0.5}$ which equates to 0.383 for the idf_{sub} calculation and 0.396 for the idf_{col} .

The performance metrics of the collection-wise idf approach are slightly better than the subchapter-wise idf. Therefore, we will focus on the collection-wise idf calculation technique in the next section, because the peak performance will be achieved by using idf_{col} . At referring to the baseline approach we mean the baseline approach using the collection-wise idf. In the following sections, we will try to improve the recommendation performance by utilizing additional parameters.

Metric	value _{sub}	value _{col}
Accuracy	0.982	0.983
Precision	0.543	0.576
Recall	0.176	0.176
F-Measure	0.266	0.27
$F_{\beta=0.5}$	0.383	0.396

Table 5.4: Performance metrics for baseline approach

5.3 Detecting joins

The prototype in our baseline approach utilizes pattern matching in combination with a list of relevant SQL keywords to detect if a keyword is contained in a slide or exercise. This is insufficient for the detection of joins that are formulated with the WHERE keyword. Therefore, we implemented a method in our prototype to recognize these joins. Figure 5.5 shows the confusion matrix of our join detection alongside the rate of change compared to the baseline approach using collection-wise idf values. The confusion matrix for the idf_{sub} values with join detection are shown in the appendix. Applying the join detection yields a positive effect on the classification results. The number of true positive predictions increased by 15.8% while the number of false positive predictions decreased by 21.4%. The join detection also has a beneficial effect on the false negative and true negative predictions although they profited percentage-wise significantly less compared to the positive predictions. This is because the number of instances in the column of negative predictions is higher than the number of instances in the second column and thus the false negative and true negative table cells are less affected percentage-wise. The performance metrics of the baseline approach with and without the join detection are shown in Figure 5.6. Each of the performance metrics increased with the activated join detection. The accuracy value increased almost negligible from 0.983 to 0.984. The recall and F-measure improved more with 0.176 to 0.204 and 0.27 to 0.312 respectively. Especially noteworthy is the increase of the precision value from 0.576 to 0.667 through enabling the join detection.

idf_{col}	pred. pos.	pred neg.
actual pos.	44 \uparrow 15.8%	172 \downarrow 3.4%
acutal neg.	22 \downarrow 21.4%	11642 \uparrow 0.052%
total	66 \pm 0	11.814 \pm 0

Table 5.5: Results of activated join detection compared with baseline approach

Metric	$\neg(\text{join detection})$	join detection
Accuracy	0.983	0.984
Precision	0.576	0.667
Recall	0.176	0.204
F-Measure	0.27	0.312
$F_{\beta=0.5}$	0.396	0.459

Table 5.6: Performance comparison with and without join detection

5.4 Clustering keywords

In our prototype we implemented the option to create a list of SQL keywords that are treated as belonging to the same cluster. If a slide or exercise contains two or more members of a cluster, we will set the $\text{tf} \cdot \text{idf}$ value of each keyword to the highest $\text{tf} \cdot \text{idf}$ value of the cluster. In our research we created and evaluated various clusters of which we will present the cluster that yields the best performance increase. Though it has to be noted that we encountered many clusters with negative effects on the performance. Because the performance of our baseline approach extended with the join detection and clustering is better with the idf_{col} , we will focus on the computations with the collection-wise idf . A strategy to find viable cluster candidates is to take a look at exercises for which there is no correct recommendation. These exercises need to be analysed in regards to which keywords they have in common with their desired recommendations. Keywords that are featured in both the exercise and its desired recommendations are then selected to be part of the cluster. This approach could not be successfully applied in our use case since clustering these shared keywords had a negative effect on other recommendations and thus decreased the performance. Therefore we followed a different approach. The clustering can be used to increase the cosine similarity between an exercise and its desired recommendation but it can also be used to decrease the similarity between an exercise and its incorrect recommendation. In the following, we will present how we used the later strategy to improve the performance of our prototype. Considering our dataset, we were able to identify which specific keywords need to be clustered together. Our selected cluster consists of the keywords $<$, $>=$ and SELECT. This process of choosing suitable keywords is manually and adapted to our specific dataset. Table 5.7 depicts the confusion matrix for the clustering approach. The clustering leads to 4.6% more true positive predictions while the false positive recommendations were lowered by 9.1%. The performance metrics are displayed in Table 5.8 alongside the comparison

of no cluster usage. The application of the cluster causes the accuracy to increase marginally from 0.983 to 0.984. More notably the precision rises from 0.667 to 0.697. The recall value increases slightly from 0.204 to 0.213. The improvement of both the recall and precision values cause the F_β value to increase from 0.459 to 0.479. Especially the improved precision and F_β metrics imply that the clustering of keywords helps the prototype to recommend useful slides to the students. We will explain in the remaining content of this section why the clustering leads to a performance increase by taking an exercise from the SQLValidator as an example.

	pred. pos.	pred neg.
actual pos.	46 \uparrow 4.6%	170 \downarrow 1.2%
acutal neg.	20 \downarrow 9.1%	11644 \uparrow 0.02%
total	66 \pm 0	11.814 \pm 0

Table 5.7: Confusion matrix of cluster application

Metric	\neg cluster	cluster
Accuracy	0.984	0.984
Precision	0.667	0.697
Recall	0.204	0.213
F-Measure	0.312	0.326
$F_{\beta=0.5}$	0.459	0.479

Table 5.8: Performance comparison with and without clustering

The improved performance due to the clustering is attributable to two more mappings between exercises and slides that are now done correctly. One of the exercises for which the prototype found the correct recommendation will be referred to as task E and is shown below:

```
SELECT job, MIN (ALL salary) AS min_salary
FROM employee
GROUP BY job;
```

The keyword analysis for task E yields the results shown in Table 5.9 with the recognized keywords SELECT, GROUP BY, GROUP, AS, MIN and ALL. The MIN and ALL keywords have the highest $tf*idf$ value with 1.217 assigned to it and therefore they are the most important keywords for this exercise.

task G				
keywords	count	tf	idf	tf*idf
select	1	1	0.087	0.087
group by	1	1	1.121	1.121
group	1	1	1.121	1.121
as	1	1	0.405	0.405
min	1	1	1.217	1.217
all	1	1	1.217	1.217

Table 5.9: Keyword analysis for task E from the SQLValidator

The recommendation of our prototype before the application of the cluster is incorrect since the recommended page is not helpful to the students. The chosen page is the twenty-fourth page of the ninth chapter "Views and Access Control". Page 24 contains information about the problems with aggregation views although task E does not feature any information about views. Hence, the recommendation of page 24 is not useful for students that are challenged by exercise E. Table 5.10 displays the keyword analysis for the page 24. The keywords WHERE and HAVING were recognized once and the keywords SELECT, GROUP BY, <, MIN, GROUP twice. The highest tf*idf values are reached by the keywords MIN at 1.556 and < with 1.352.

Views and Access Control
Updates via Views

Problems with Aggregation Views /2

- After simple syntactic transformation:

```

select Color
from WINES
where min(Vintage) < 1995
group by Color

```
- No syntactic correct SQL-query – correct would be:

```

select Color
from WINES
group by Color
having min(Vintage) < 1995

```

Saake
Database Concepts
Last Edited: June 11, 2020
9–24

Figure 5.1: Incorrect recommendation of page 24 from chapter nine to task E before clustering

Chapter: 9, Page: 24, Cosine: 0.682				
keywords	count	tf	idf	tf*idf
select	2	1	0.347	0.347
where	1	0.5	0.484	0.242
group by	2	1	1.109	1.109
group	2	1	1.051	1.051
having	1	0.5	1.301	0.651
<	2	1	1.352	1.352
min	2	1	1.556	1.556

Table 5.10: Keyword analysis of the incorrectly referred page 24 from chapter nine

The recommendation to task E should contain information as to how the GROUP BY keyword can be used to aggregate data. Instead of recommending page 24 of the ninth chapter, the independent labelers chose page 61 of the sixth chapter, displayed in Figure 5.2 as a good fit for task E since it visualizes the process of using the GROUP BY clause. The keyword analysis for our desired recommendation is shown in Table 5.11. Page 61 contains the three SQL keywords AND, GROUP BY and GROUP once with the GROUP BY keyword reaching a tf*idf of 1.109 and the GROUP clause following at 1.051. The comparison between Table 5.10 and Table 5.11 shows that the cosine similarity of page 61 with 0.625 is lower than the cosine value of 0.682 from the current recommendation. In order to change the recommendation from page 24 to page 61, we need to influence the cosine similarity between task E and the slides by creating a suitable cluster. At first we tried to create clusters of keywords that are both contained in task E and our desired recommendation of page 61. Unfortunately, every cluster that used this approach lead to a performance decrease and therefore a different strategy was necessary in order to establish a correct mapping for exercise E. Instead of increasing the cosine similarity of our desired recommendation, we can also decrease the cosine similarity of the current recommendation by utilizing a cluster. Page 24 contains the < and SELECT keywords with the SELECT clause being also shared with task E. We have a cluster in use that contains the SELECT and < statements and thus changes the tf*idf values of page 24. The performance evaluation of page 24 is shown in 5.12 with the SELECT keyword having a tf*idf value at 1.352 instead of 0.347. The increased tf*idf value causes the similarity between task E and page 24 to shrink and thus the new cosine value equals 0.625. The cosine similarity of page 61 does not change in respect to exercise E because our desired recommendation does not share any keywords with the cluster. The unchanged similarity of 0.64 is sufficient in order to be chosen for recommendation in task E since the former cosine value of page 24 decreased. Our cluster contains three keywords in total with the >= clause not being mentioned yet. In our research we observed a performance decrease at using a cluster that only contains the SELECT and < keywords. We believe that using the cluster without the >= yields a side effect to the other exercises which is why we chose to include >= in our cluster. There is more information about side effects due to clustering in Section 5.4.

The Relational Query Language SQL
Aggregation and Grouping

Grouping: Step 2

- **group by A, B**

A	B	C	D
1	2	3	4
1	2	4	5
2	3	3	4
3	3	4	5
3	3	6	7

➡

A	B	N	
		C	D
1	2	3	4
		4	5
2	3	3	4
3	3	4	5
		6	7

Saake
Database Concepts
Last Edited: June 11, 2020
6-61

Figure 5.2: Page 61 of the sixth chapter which should be chosen for recommendation

Chapter: 6, Page: 61, Cosine: 0.64				
keywords	count	tf	idf	tf*idf
GROUP BY	1	1	1.109	1.109
GROUP	1	1	1.051	1.051
AND	2	1	0.499	0.499

Table 5.11: Keyword analysis for page 61 from chapter six

Chapter: 9, Page: 24, Cosine: 0.625				
keywords	count	tf	idf	tf*idf
select	2	1	0.347	1.352
where	1	0.5	0.484	0.242
group by	2	1	1.109	1.109
group	2	1	1.051	1.051
having	1	0.5	1.301	0.651
<	2	1	1.352	1.352
min	2	1	1.556	1.556

Table 5.12: Keyword analysis for page 24 from chapter nine after clustering

5.5 Variations in the cosine cutoff

The baseline approach of our prototype only uses the pages with the best cosine values for recommendation. This might be insufficient since there could be several useful pages achieving high cosine values. Solely considering the page with the best value prevents the other high ranking pages from being recommended. We implemented the option to extend the recommendation to pages that are within a certain value range of the highest cosine value. The user is able to define the value range that a page has to be within by selecting a fraction of the highest cosine value that each page has to reach. In this section, we will present the change in performance by further extending the baseline approach with a cosine-cutoff at the values 0.25, 0.5 and 0.75. The baseline approach without the cosine-cutoff option is equal to a cosine-cutoff at 1.0 because there are no pages recommended that have a lower cosine similarity than the best page. We will evaluate the confusion matrices for each cosine-cutoff step while using the collection-wise idf computation since this will lead to the best results. However, at the end of this section we present a comparison between performance metrics for these cosine-cutoffs alongside the performance for the subchapter-wise idf calculation. Table 5.13 shows the confusion matrix for the baseline approach combined with a cosine-cutoff at 0.75. In comparison to the baseline approach without a modified cosine-cutoff it shows that overall more pages are now recommended. The number of true positives increased from 46 to 112 while the false positives have more than seven folded with 20 compared to 171.

cutoff = 0.75	pred. pos.	pred neg.
actual pos.	112↑ 143.5%	104↓ 38.82%
acutal neg.	171↑ 755%	11493↓ 1.3%
total	283 ↑ 328.79%	11.597 ↓ 0.91%

Table 5.13: Confusion matrix for cosine-cutoff = 0.75

cutoff = 0.5	pred. pos.	pred neg.
actual pos.	138 ↑ 200%	78 ↓ 54.12%
acutal neg.	522 ↑ 2510%	11142 ↓ 4.31%
total	660 ↑ 900%	11.220 ↓ 5.03%

Table 5.14: Confusion matrix for cosine-cutoff = 0.5

cutoff = 0.25	pred. pos.	pred neg.
actual pos.	180 ↑ 291.3%	36 ↓ 78.82%
actual neg.	1341 ↑ 6605%	10323 ↓ 11.35%
total	1521 ↑ 2204.55%	10.359 ↓ 12.32%

Table 5.15: Confusion matrix for cosine-cutoff = 0.25

The Tables 5.14 and 5.15 display the confusion matrix regarding a cosine-cutoff at 0.5 and 0.25. The comparison of these tables with Table 5.13 reveals again that a lower cosine-cutoff leads to more pages being recommended overall. A cosine-cutoff at 0.75 leads to 283 recommendations in Table 5.13 while a cutoff at 0.5 results in 660 recommendations displayed in Table 5.14 and a threshold at 0.25 leads to 1520 recommendations shown in Table 5.15. It has to be noted that most of the additional recommended pages are actually false positive and thus are not suited to support the students at their task. A change of the cosine-cutoff from 0.75 to 0.25 yields a disproportionate increase of false positives that raised by 683.63% while true positives just increased by 60.7%. Table 5.16 displays the comparison of performance metrics for each of the previously presented cosine-cutoff values alongside the baseline approach that uses a cosine-cutoff at 1.0. Considering the accuracy metric for each column of the table it shows that a higher cosine-cutoff results in a higher accuracy value with the threshold at 1.0 having the highest accuracy of 0.984. A cutoff at 0.25 results in the lowest accuracy with 0.884. A similar effect is observable with the precision value that is declining as the cosine-cutoff value decreases. The highest precision is reached with the baseline approach at a value of 0.697 while a cosine-cutoff at 0.25 has the lowest precision value of 0.118. In Section 2.6.3 we stated that recall and precision values are inversely correlated. This is observable in Table 5.16 since the precision value declines at lowering the cosine-cutoff while the recall value increases. The reason for this is that the recall stands for the fraction of relevant instances that were retrieved. By making the recommendation requirements less strict more pages are being recommended and some of these additional selected pages are relevant hence the recall value increases. But as more pages, that are not labeled as useful for the students, are chosen to be recommended the fraction of actually relevant instances among the instances predicted as recommendable shrinks and thus the precision decreases. Since our focus is to achieve a high precision value, we will set the cosine-cutoff at 1.0 because otherwise the precision declines.

Metric	cutoff _{1.0} \wedge idf_{sub}	cutoff _{1.0}	cutoff _{0.75}	cutoff _{0.5}	cutoff _{0.25}
Accuracy	0.984	0.984	0.977	0.949	0.884
Precision	0.657	0.697	0.396	0.209	0.118
Recall	0.213	0.213	0.519	0.639	0.833
F-Measure	0.322	0.326	0.449	0.315	0.207
$F_{\beta=0.5}$	0.464	0.479	0.416	0.242	0.142

Table 5.16: Performance metrics for evaluated cosine-cutoff rates

5.6 Minimal cosine value

During our research, we have noticed that there are some exercises for which even the best slides have a rather low cosine similarity. These low cosine similarities indicate that there is no suitable page to recommend to the students. Suppose our prototype computes a recommendation for an exercise and it turns out that the best page for this exercise yields a cosine value of 0.1. The value range of cosine values is in the interval of $[0 - 1]$ with zero meaning no similarity and 1.0 being almost

identical. Considering a cosine value 0.1 it is likely that this recommendation is not very helpful to the students since the similarity between exercise and the slide is rather low. Instead of then recommending the page with the highest cosine value, which in this case rather means the least bad page, we propose to not recommend any page for this particular exercise. While not recommending any slide is not helpful for the student, recommending a page that is not explaining the topic of the exercise properly creates a cost for the students in terms of confusion and wasted time. Therefore, we included the option to set a minimal cosine similarity that has to be reached for any page in order to be recommended. In the following, we will present in Table 5.17 the results of further extending the baseline approach with a minimal cosine value at 0.2, 0.4, 0.6. The second column of the table contains the performance for the baseline approach using idf_{col} with the extensions introduced in Sections 5.3, 5.4 and 5.5. The baseline approach uses a minimal cosine value of zero because there is no threshold implemented that restricts pages with low cosine values from being recommended. The third column displays the performance values for a minimal cosine value of 0.2. A comparison between the baseline approach and a minimal cosine value of 0.2 shows that the minimal cosine value has a small positive effect on the performance. The precision, F-measure and $F_{\beta=0.5}$ slightly increase while the recall value remains unchanged. This implies that the minimal cosine value filtered out slides that were incorrectly recommended. Using a minimal cosine value of 0.4 increases the performance further with a precision value of 0.767 and a $F_{\beta=0.5}$ of 0.505. The recall value is still unchanged which means that by setting the minimal cosine value to 0.4, there are just incorrect recommendations being filtered out. The fifth column displays the performance of setting the cosine value to 0.6. There is a slight decrease in all metrics compared to the previous column. This is because increasing the cosine value eventually leads to correct recommendations being filtered out which then leads to performance values. Since our focus is on maximizing the F_{β} and precision values, we should select a minimal cosine value of 0.4 regarding our prototype.

Metric	baseline	cos-min _{0.2}	cos-min _{0.4}	cos-min _{0.6}
Accuracy	0.984	0.984	0.985	0.984
Precision	0.697	0.708	0.767	0.763
Recall	0.213	0.213	0.213	0.208
F-Measure	0.326	0.327	0.333	0.327
$F_{\beta=0.5}$	0.479	0.483	0.505	0.498

Table 5.17: Performance metrics for evaluated minimal cosine values

5.7 Discussion

The evaluation has shown that a reasonable mapping between slides of the lecture and SQL exercises can be achieved. Our prototype is able to provide meaningful feedback for many exercises and seldom recommends slides that are not meant to be recommended. Extending the baseline approach with additional techniques enabled us to increase the performance of the prototype significantly especially considering

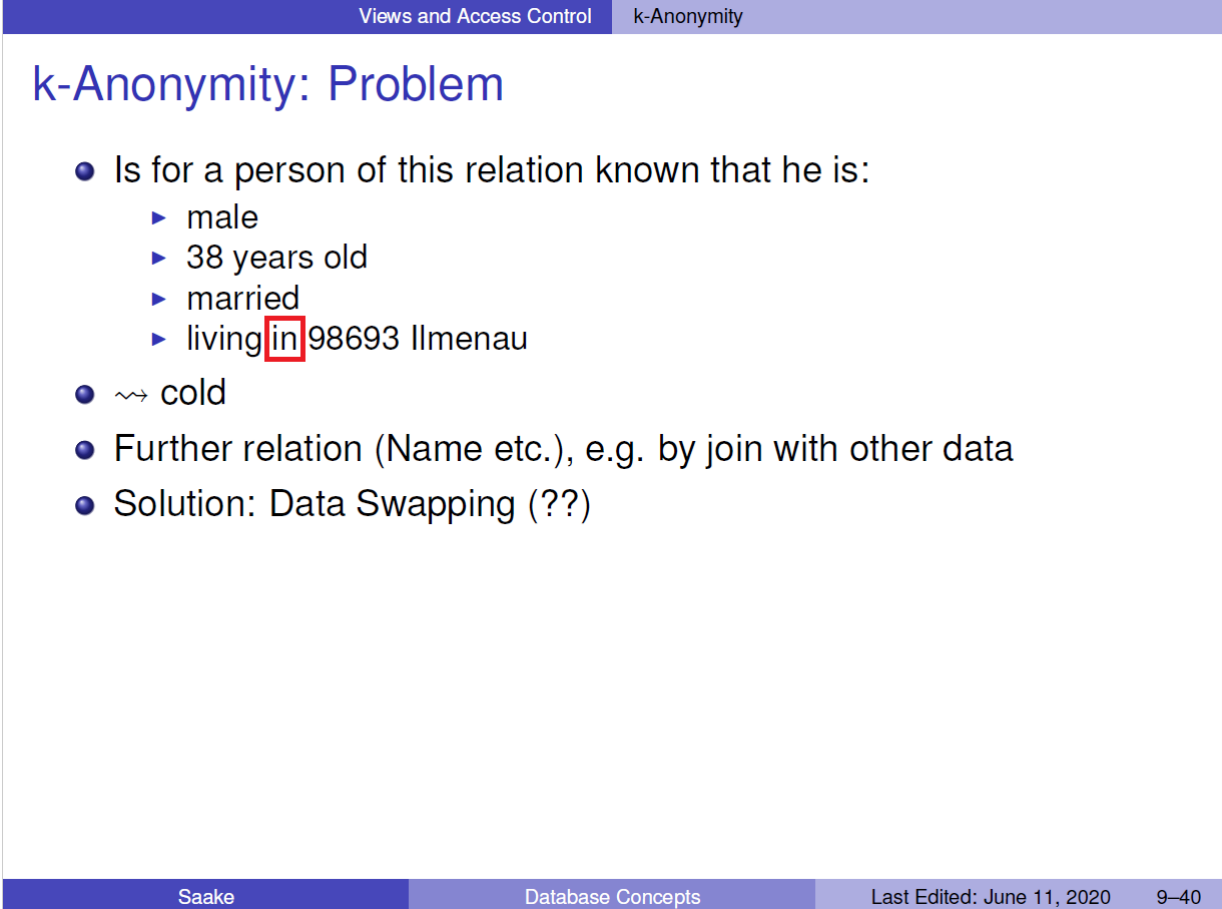
the precision value. Hence, the probability of the prototype confusing the students by recommending slides that are not relevant to the lecture has been reduced. The performance evaluation has revealed that we achieved our goals at providing a method to support the learning experience of students that are challenged by understanding the SQL programming language. However, there are still some problems concerning the recommendation process and in the following we will explain what hurdles we have encountered in our experiments. Some SQL keywords are regularly used in the English language without any SQL context. In Section 5.7.1, we explain the problems our prototype has at recognizing these keywords. Clustering keywords together improved the performance of our prototype. However, there are some drawbacks to using a cluster. Section 5.7.2 will address those. During the analysis of our recommendations, we have discovered that pages containing a high number of keywords are seldom recommended. The reason for this behaviour is presented in Section 5.7.3.

5.7.1 Usage of keywords in the English language

One problem we encountered during our research was that some SQL keywords such as AND, IN or AS are regularly used in the English language without any SQL context. Since the lecture slides are written in English, problems arise when counting the number of times these keywords are used in a SQL environment. This is especially challenging for our recommendation process if the encountered keyword appears rarely in the corpus as it would receive a high idf weight leading to a great influence in deciding the topic of the page. In case of a page populated with many SQL keywords the negative effect of one incorrectly recognized keyword might be mitigated by the $tf*idf$ values of the other keywords. If the page only has a small number of keywords, it might happen that a word like IN or AND not used in any SQL context will mislead the recommendation process for this page. Figure 5.3 displays page 40 of the ninth chapter which contains the IN keyword although it is not used in any SQL context. Table 5.18 presents the keyword analysis of this page. The keywords JOIN and IN are recognized and the $tf*idf$ value for both is shown in the fifth column with 0.699. By incorrectly recognizing the IN keyword in page 40 of the ninth chapter, it might result in linking this page to an exercise that requires the students to understand the correct usage of the IN keyword although the page does not hold any information about the IN keyword.

Chapter: 9, Page: 40				
keywords	count	tf	idf	$tf*idf$
in	1	1	0.699	0.699
join	1	1	0.699	0.699

Table 5.18: $Tf*idf$ analysis for page 40 from chapter nine which features a SQL keyword in a non SQL context



Views and Access Control k-Anonymity

k-Anonymity: Problem

- Is for a person of this relation known that he is:
 - ▶ male
 - ▶ 38 years old
 - ▶ married
 - ▶ living in 98693 Ilmenau
- \rightsquigarrow cold
- Further relation (Name etc.), e.g. by join with other data
- Solution: Data Swapping (??)

Saake Database Concepts Last Edited: June 11, 2020 9-40

Figure 5.3: Slide 40 from chapter nine that contains a SQL keyword not used in a SQL context

5.7.2 Clustering keywords

In Section 5.4 we were able to improve the performance by using a cluster of keywords. We chose the keywords for this cluster to specifically target the recommendations for particular exercises. We think that the clustering option can be effective but needs to be used with caution. Even though the cluster list is targeted at a particular exercise, it will be applied to all of the other exercises as well. This might lead to side effects concerning other recommendations for instance correct mappings of exercises to pages could change to incorrect mappings or cosine similarities of correct mappings could decrease. The cluster usage also might be counterintuitive. It could happen that by the cluster application the cosine similarity of a desired page is not changed. Instead, the cosine similarity of the currently best recommendation decreased which could have the effect of recommending the desired page although its cosine similarity did not change much. The user might not be aware of these side effects because there are many recommendations which are tedious to manually check for side effects upon.

5.7.3 Pages with many keywords

In our research we have encountered a problem at recommending pages that contain a high number of SQL keywords to exercises. The reason for this lies in the formula for the cosine similarity shown in Section 2.4. The dot product of an exercises vector and a pages vector is divided by the magnitude of both vectors. A higher number of keywords in the vector of the page leads to an increase of the term $\sqrt{\sum_{i=1}^N b_i^2}$ that is part of the denominator and thus reduces the cosine similarity. We will visualize this behaviour by taking page 20 from the sixth chapter as an example in Figure 5.5. Page 20 of the sixth chapter covers multiple SQL concepts such as the WHERE clause, several comparison operators and logical connections. The result of the keyword analysis is displayed in Table 5.19. Twelve different keywords have been recognized with the AND, WHERE and OR keywords appearing twice. We think that this page is a useful recommendation for multiple exercises of the SQLValidator, one of those is the following SQL query which is a solution for task 2.a:

```
SELECT name, salary
FROM employee
WHERE salary > 6450;
```

This task encompasses a query using the WHERE clause in addition to the comparison operator >. Both of these concepts are shown in the presented page. By using the cosine similarity we were not able to connect the exercise to this slide because the magnitude of the slides vector is too high due to the high number of featured keywords which causes the denominator to increase and finally the cosine similarity to decrease. Instead of the page that we would like to recommend to the exercise, our prototype calculates page 19 of the second chapter as the best recommendation as shown in Table 5.19. Unfortunately, this page is about the relational operators and thus does not represent the concept of our task at all. From the perspective of the keyword analysis, shown in Table 5.19, the cosine similarity in regards to the exercise is very high with 0.962 although this page only shares the > keyword with the exercise.

Chapter: 2, Page: 19, Cosine: 0.927				
keywords	count	tf	idf	tf*idf
in	1	1	0.273	0.273
>	1	1	1.477	1.477
as	1	1	0.316	0.316

Table 5.19: Tf*idf analysis for page 19 from chapter two which is incorrectly chosen as recommendable for task 2.a

Chapter: 6, Page: 20, Cosine: 0.393				
Keywords	Count	tf	idf	tf*idf
select	1	0.5	0.347	0.174
where	2	1	0.484	0.484
and	2	1	0.449	0.449
or	2	1	0.857	0.857
not	1	0.5	0.847	0.429
in	1	0.5	0.273	0.137
between	1	0.5	1.778	0.889
>	1	0.5	1.477	0.739
<	1	0.5	1.352	0.676
>=	1	0.5	1.778	0.889
<=	1	0.5	1.556	0.778
=	1	0.5	0.548	0.274

Table 5.20: Tf*idf analysis for page 20 from chapter six which should be chosen for recommendation for task 2.a

Relational Databases – Data as Tables
Basic Operations: The Relational Algebra

Selection σ

- **Selection:** Choose rows in a table based on a selection predicate

$$\sigma_{\text{Vintage} > 2000}(\text{WINES})$$

WineID	Name	Color	Vintage	Vineyard
2168	Creek Shiraz	Red	2003	Creek
3456	Zinfandel	Red	2004	Helena
2171	Pinot Noir	Red	2001	Creek
4961	Chardonnay	White	2002	Bighorn

Saake
Database Concepts
Last Edited: May 2, 2020
2-19

Figure 5.4: Page 19 of the second chapter as an example for advantages that slides with few keywords have at recommendation

The Relational Query Language SQL

The SFW Block in Detail

The where Clause

```
select ...from ...  
where condition
```

- Forms of the condition:
 - ▶ Comparing an attribute with a constant:
$$\text{attribute } \theta \text{ constant}$$

possible comparison symbols θ depend on the domain; e.g., =, <>, >, <, >= or <=.
 - ▶ Comparison between two attributes with compatible domains:
$$\text{attribute1 } \theta \text{ attribute2}$$
 - ▶ Logical *connectors* **or**, **and** and **not**

Saake

Database Concepts

Last Edited: June 11, 2020

6-20

Figure 5.5: Page 20 of chapter six which is not chosen for recommendation due to too many keywords

6. Related work

This section is devoted to providing the reader with an overview of several already existing SQL processing tools and to what extent the SQLValidator with its improved feedback through recommendation differentiates from these frameworks. An educational tool developed by the Arizona State University enables the students to learn about relational query languages by letting the students solve their homework assignments using the tool [Dietrich, 1993]. However, the students receive their feedback directly from the interpreter and thus we assume the feedback to be too technical to be understood by class participants with minor knowledge about SQL. In comparison, our recommendation points the students to the material of the lecture that explains the correct usage of SQL concepts and hence the students quickly understand which command they have used incorrectly. Another tool to support students at learning SQL is the SQL-Tutor developed by the University of Canterbury [Mitrović, 1998] and used at many universities around the globe. The tool prompts students to solve SQL related exercises and while doing so it provides different stages of feedback to the students. The feedback although is restricted to the task at hand and is not utilizing the learning materials of the course like the SQLValidator does. Finally, during our research for related work we have found the SQL Tester [Kleerekoper and Schofield, 2018] tool that is employed at the University of Manchester. The SQL Tester is used to assess and grade the SQL knowledge of students during a 50 minute test that consists of ten SQL exercises. However, the students only received the regular error messages from the RDBMS as feedback. We are not aware of any research that is trying to improve the SQL learning experience by automatically connecting exercises to the material of the lecture.

7. Conclusion

The motivation for this bachelor thesis was to find a method to support students that learn SQL using the SQLValidator. Especially students that are facing problems at solving SQL related exercises are the focus of our effort. Students that fail to solve particular exercises need to be given a hint as to which SQL domain they lack knowledge in. To this end, we have implemented a prototype that is able to automatically map suitable slides of the lecture to SQL tasks so that students which need additional support at solving the exercise receive a recommendation which pages of the lecture they should take a look at. Our work at creating the prototype started with converting the pdf that contains the slides of the lecture into a string which then can be further analysed. For each page, SQL keywords are detected and weighted based on how well the keywords can be used to distinguish one page from another. In the next step, it was necessary for us to apply the analysis for the SQL exercises since those have to be later compared with the slides of the lecture. The SQL exercises are queried from a database and analysed analogously to the slides of the lecture. After the $tf*idf$ values for both exercises and the slides of the lecture are derived, our prototype computes the cosine similarity for each entry of the cartesian product between exercises and slides. The similarity values are then compared and the pages with the highest cosine values are selected to be referred to students. This process has been improved through the application of several optimization techniques. Our prototype offers the user to set a minimal cosine value that has to be reached in order for slides to be recommended to students. The recommendation can be extended with slides that are reaching a selected fraction of the best slides cosine value. Joins that are formulated using the WHERE keyword can be transformed into regular joins and multiple keywords can be clustered together with each member of the cluster getting assigned the best $tf*idf$ value of all members. Our prototype has shown that a mapping between SQL exercises and the lecture's slides can be used in connection with the cosine similarity for creating reasonable recommendations. The performance has been increased significantly by the optimization techniques and after enabling the optimizations, we can confidently state that the performance of our prototype justifies the approach of recommending slides of the lecture to certain

SQL exercises based on the cosine similarity. The best performance of our prototype reaches a precision value of 0.767 and a $F_{\beta=0.5}$ value of 0.505.

7.1 Future work

In this section, we will present our ideas regarding possible future improvements for our prototype.

Automated test runs

During our research we had to manually start each test run. This is tedious because there are many test runs necessary to evaluate all the different possible configurations our prototype can be used with. Especially the option to cluster several keywords together offers a high amount of possible combinations that would be overwhelming to test manually. In order to reduce the time invested in manually executing the configuration and starting the program, we propose to extend the prototype with a framework that automatically tests several configurations and stores the result for later analysis.

Recognize SQL concepts

Our prototype relies on purely recognizing SQL keywords with the extension of detecting joins that are created with the WHERE keyword. Whenever our prototype encounters a join formulated with the WHERE keyword it will ignore the amount of occurrences for the WHERE and = keywords and instead increment the counter of the JOIN keyword. Analog to the JOIN detection there are more SQL concepts that are not recognized well by simply increment the counter of its keywords. For instance, the detection of nested queries might be beneficial to the performance of our prototype. We would like to add concepts such as nested queries to the keyword list of the prototype. Each time a nested query occurs the number of occurrences for the SELECT keyword should be ignored and instead the counter for nested queries needs to be incremented.

Further text preprocessing

A very popular technique in text classification is called "stemming". Through the application of stemming, words with the same stem are treated as if they were identical. In our case, this would lead to words such as "join" and "joins" both be recognized as "join". Since our prototype uses a keyword list consisting of SQL keywords the appearance of "joins" would increase the JOIN counter. Utilizing stemming could improve the performance since a slide from the lecture might be explaining the concept of "joins" while rarely mentioning the JOIN keyword explicitly.

Appendix

	pred. pos.	pred neg.
actual pos.	44 \uparrow 15.79%	172 \downarrow 3.37%
acutal neg.	26 \downarrow 18.75%	11638 \uparrow 1.46%
total	70 \pm 0	11.810 \pm 0

Table A.1: Results of activated join detection compared with baseline approach for idf_{sub}

Metric	\neg (join detection)	join detection
Accuracy	0.982	0.983
Precision	0.543	0.629
Recall	0.176	0.204
F-Measure	0.266	0.308
$F_{\beta=0.5}$	0.383	0.444

Table A.2: Comparison of metrics with and without join detection for idf_{sub}

Bibliography

- Michael Buckland and Fredric Gey. The relationship between recall and precision. *Journal of The American Society For Information Science and Technology*, 45(1): 12–19, 1994. (cited on Page 8)
- Stefano Ceri, Alessandro Bozzon, Marco Brambilla, Emanuele Della Valle, Piero Fraternali, and Silvia Quarteroni. An introduction to information retrieval. In *Web information retrieval*. Springer, 2013. (cited on Page 8)
- Donald D Chamberlin and Raymond F Boyce. SEQUEL: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264, 1974. (cited on Page 10)
- Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 26(1), 1970. (cited on Page 9)
- Suzanne Wagner Dietrich. An educational tool for formal relational database query languages. *Computer Science Education*, 4(2):157–184, 1993. (cited on Page 51)
- Anthony Kleerekoper and Andrew Schofield. SQL Tester: An online SQL assessment tool and its impact. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, page 87–92. Association for Computing Machinery, 2018. (cited on Page 51)
- Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2nd edition, 2014. (cited on Page 4 and 6)
- Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2nd edition, 2008. (cited on Page 4 and 5)
- Antonija Mitrović. Experiences in implementing constraint-based modeling in SQL-Tutor. In Barry P. Goettl, Henry M. Halff, Carol L. Redfield, and Valerie J. Shute, editors, *Intelligent Tutoring Systems*, pages 414–423, 1998. (cited on Page 51)
- Victor Obionwu, David Broneske, Anja Hawlitschek, Veit Köppen, and Gunter Saake. SQLValidator – An Online Student Playground to Learn SQL. *Datenbank. Spektrum.*, pages 1–9, 2021. (cited on Page 1)
- Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation - J DOC*, 60:503–520, 2004. (cited on Page 6)

- Gunther Saake, Kai-Uwe Sattler, and Andreas Heuer. *Datenbanken: Konzepte und Sprachen*. MITP-Verlags GmbH & Co. KG, 2018. (cited on Page 10, 11, and 12)
- Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988. (cited on Page 4 and 5)
- Claude Sammut and Geoffrey I. Webb. *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, 1st edition, 2011. (cited on Page 9)
- Yutaka Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 2007. (cited on Page 9)
- Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, 18(3):491–504, 2014. (cited on Page 6)
- Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2005. (cited on Page 7)

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

Magdeburg, 29th July 2021