An Indexing Learning Tool in Relational Databases

Marko Vještica, Slavica Kordić, Vladimir Dimitrieski, Milan Čeliković, Ivan Luković
University of Novi Sad, Faculty of Technical Sciences
Department of Computing and Control Engineering
Trg Dositeja Obradovića 6, 21000, Novi Sad, Serbia
{marko.vjestica, slavica, dimitrieski, milancel, ivan}@uns.ac.rs

***Abstract**. There are multiple ways to optimize a query execution time in database systems, but one of the most usual ways is to use indexing techniques. In database courses at University of Novi Sad, Faculty of Technical Sciences, we teach students about these techniques and we want to improve lab sessions by providing them with as much knowledge in short amount of time. We have created Indexing Learning Tool (ILT) to help our students learn about indices and their usage in database systems. With this tool students can make fast changes in data structures and try different indexing techniques on different use cases. ILT provides new insights into the usage of indexing techniques, their advantages and disadvantages.*

Keywords. Learning tool, Database optimization, Indexing techniques, Data warehouse, Relational database

1 Introduction

A database optimization is important field of study, because many organizations need to gather reports fast. Execution time of queries need to be short and transactions need to be fast, because they are crucial to have successful environment in companies. If companies have a Data Warehouse system, it is more important to optimize a query execution time than the transaction response time (Chaudhuri & Dayal, 1997). To improve the query execution time, different techniques can be used like a parallel query processing, data partitions or query transformations, but most usual are indexing techniques and materialized views. They are both useful, especially in Data Warehouse systems, because these systems are mainly read-only.

At University of Novi Sad, Faculty of Technical Sciences, undergraduate students of Control and Computing Engineering learn about basics of database systems, database modeling and *SQL*. In “Data Warehouse Systems” and “Database Management Systems” courses on master studies, they also learn about database optimization using indexing techniques and materialized views. As a part of courses’ labs, students try to improve database performance by using these techniques. As the time is limited to experiment with these techniques, our goal was to help students by creating a tool that will enable fast changes on data structures and experiments. A purpose of that tool is not to teach students *SQL*, but to help them learn when to use indices.

Not all indices are useful, some of them could even prolong a query execution time in specific use cases. There is an index selection problem which presents the selection of appropriate indices in Data Warehouse systems (Golfarelli & Rizzi, 2018). There are many existing solutions and heuristics that we want to test with students. Our goal is to improve studies in the field of the index selection problem by using the interactive tool which can support different use cases and analysis.

In the work of (Sadiq et al., 2004), they stated that the best way to learn *SQL* is to have well directed practical classes and to let students learn from their mistakes. The same could be stated about learning indexing techniques usage. According to (Kenny & Pahl, 2009), students should have control over their learning experience, because it will encourage their sense of responsibility and increase their motivation. We want to motivate students to use the tool so they can research and discover advantages and disadvantages of indexing techniques and learn from their mistakes.

Professional database tools are not meant for educational purposes, mainly because of their complexity (Grillenberger & Brinda, 2012). We only use a part of their functionalities and adapt them for learning. This tool is mainly designed for our students, but it could also be used by anyone who wants to learn more about index selection problem.

In this paper, we will present Indexing Learning Tool (ILT) and use cases intended to be used by students in labs. Apart from Introduction, this paper is organized as follows. Section 2 describes related work of different database learning tools and presents the index selection problem. Section 3 presents the ILT architecture with all its modules. Section 4 describes databases that are necessary for our tool to store test and result data. Section 5 presents use cases in which we measured a query execution time provided by

indices and their memory usage. Section 6 contains conclusions and future work guidelines.

2 Related Work

Many educational database tools are designed for users to learn *SQL*. One of the motivations to create such tools is their adaptation for educational purposes and simplifying usage, like it is in (Grillenberger & Brinda, 2012). They developed a web-based learning environment for *SQL* and adapt it for students of secondary computing schools. Our motivation is similar, but we wanted to create the tool for education of indexing techniques in relational databases. In this section, we will present related work and tools for educational purpose in database fields that are already implemented.

Some tools support automated grading of SQL tasks. (Kleiner, Tebbe & Heine, 2013) implemented a software system that automatically determines a score for every select statement, based on various parameters. This software system increased students' motivation to analyze their errors and learn more about SQL. There is also the SQL Tester tool that is designed for students to learn SQL through tests and provides them with a score, which increased student achievements (Kleerekoper & Schofield, 2018). Authors' motivation to create that tool was to reduce time during lab sessions and motivate students to learn more, which is also something that motivate us.

In the paper (Kenny & Pahl, 2009), created a web-based e-learning environment for tutoring *SQL*, with adaptive feedback and guidance based on the student progress, which increased students' performance in exams. We plan to add a module that will help our students in learning about the index usage by providing hints and guidelines based on the existing heuristics. For example, a user wants to test an execution of a query with *MAX* aggregation function on an attribute with values that are all unique. According to (O'Neil & Quass, 1997), a B-tree index should provide the best performance in that use case. ILT could advice students to try creating B-tree index on the required attribute and execute that query. This could help students to learn faster, especially if they are not sure how to optimize the query execution. Similar was done in SQL Tutor, created by (Agha, Jarghon & Abu-Naser, 2018). This tool is created for students of *SQL* programming and it provides customized hints for each student. SQL Tutor also contains multimedia tutorials, which was also included in SQLator (Sadiq et al., 2004), a tool that allows a user to evaluate correctnes of queries that she or he created. Currently, ILT doesn't include any tutorial about indexing techniques, because students already know basics and the idea is that they research new knowledge.

Not all tools support SQL learning only as there are tools that provide learning in other database fields, like a web-based tool created by (Kung & Tung, 2010) that helped students in understanding ER and relational data modeling. However, as far as we know, there is no tool implemented for education of indexing techniques usage. Our tool can be used by anyone who knows basics about indexing techniques but wants to improve their knowledge about index selection problem. We are using the tool with our students in practical classes to apply their knowledge from theoretical classes.

There are many heuristics that users could follow to improve query performance and memory usage in different use cases. In the work of (Jurgens & Lenz, 2001), it is stated that there are nine different parameters that could influence a query execution time. Parameters like the number of records in tables, the number of attributes of tables or the number of different values of attributes could be variable using ILT. In section 5, we presented use cases in which we were changing the number of records and measured queries execution time so we can analyze an influence of this variable parameter.

3 ILT Architecture

ILT is made to help students learn indexing techniques in relational databases. The idea behind the tool is not to teach students SQL syntax, but to teach them how to use indices in different use cases. Students can use ILT to create a database schema and to insert specific data in the database for testing purpose. They can create and execute different queries on test data, with or without help of indices. The performance metrics of these queries is then stored in a database together with indices memory usage. These results can be visualized and presented to the students, so they can learn about advantages and disadvantages of using different indices when executing queries. In Fig. 1 we present an architecture of ILT. In this section, we will describe every module of its architecture. Module names are given in bold together with the databases, a user and the tool, so it could be easier to follow on the figure.

3.1 ILT modules

By the notion of **User** in this paper we refer to a student, because **ILT** is mainly created to fulfill needs of students in our database courses. Our tool comprises seven modules and a repository with two databases. First, we will provide a brief description of all modules and databases, and afterwards we will present them in more details.

Test Database contains tables and indices that are needed for testing. These data structures will be used in many queries, which execution time will be measured. **Result Database** stores execution time of queries and memory usage of data structures that are in **Test Database**. **Data Structure Controller** is used by the user to create tables and indices in **Test Database** and **Record Controller** populates these tables and indices with records. The user can insert data by adding

records one by one or using **Record Generator** to randomly insert many records in the database. **Data Structure Information Gatherer** gathers information about tables and indices and saves them into **Result Database**. The user can create a query with **Query Controller**, which will be sent to **Test Database**. Its execution time is measured, and execution plan gathered by **Query Execution Plan Controller and Viewer**. They are presented to the user and sent to **Result Database**. **Result Visualizer** can create different charts using results stored in **Result Database** to help the user analyse them in detail. In the continuous of this subsection, a full description of these modules will be presented and in the Section 4, we will present the databases schemas.

The **Data Structure Controller** module contains two controllers: **Table and Constraint Controller** and **Index Controller**. They are used to update tables and their constraints and indices. The user initializes a change of database schema using these controllers. With **Table and Constraint Controller**, the user can choose to create a new table and define attributes and constraints of the table, or to change or drop existing tables. To create a new index, the user can specify index type and what she or he wants to be indexed, or the user can change or drop existing indices using **Index Controller**. Right now, our tool supports B-tree, bitmap and function-based indices. After the user choses data structure changes, they will be sent to **Test Database** as SQL *Create*, *Alter* or *Drop* statements. After database schema changes are sent**,** the property changes like table name, index name, index type, what is indexed, attribute and constraint properties will be sent to **Data Structure Information Gatherer**. The gatherer will later send these property changes to **Result Database**.

The **Record Controller** module is used by the user to insert new data or change existing ones in **Test Database**. There could already exist data that the user inserted before or that are data that exists as examples in ILT, so the user could immediately test some queries. The user needs to decide if she or he wants to

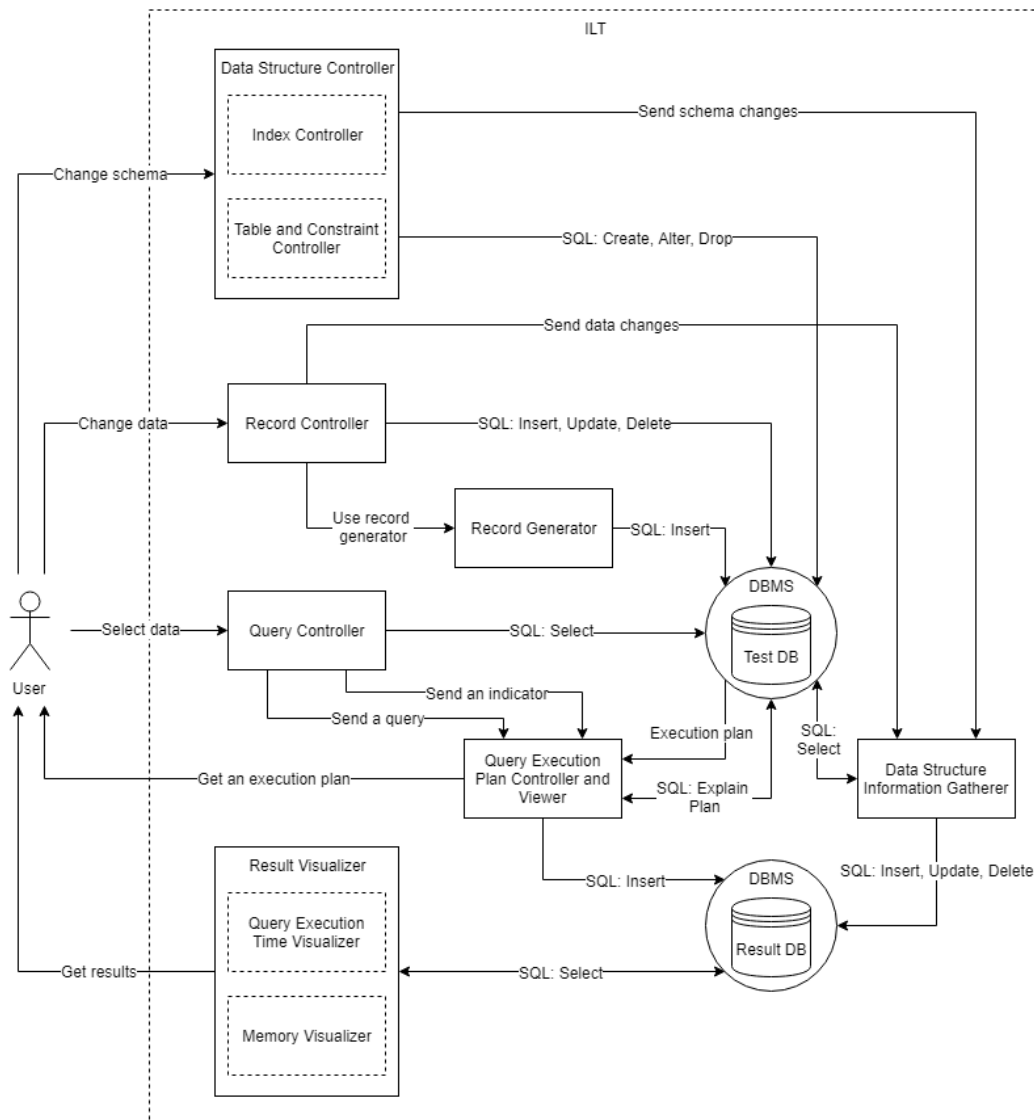

Figure 1. The ILT architecture

define specific data changes, after the SQL *Insert*, *Update* or *Delete* statements will be sent to **Test Database**, or the user wants to use **Record Generator** to randomly generate records to the same database. The changes like the number of records in the tables or the number of unique values of attributes will be sent to **Data Structure Information Gatherer**, after data changes are sent to **Test Database**.

The **Record Generator** module uses a code generator to generate SQL scripts to insert data into **Test Database**. The user can choose how many records will be generated, what will be the range of generated values and how many unique values will be generated within any attribute. The user needs to have a control over the value generation, because different types of indices could be appropriate to use in different scenarios, which depends of the values of the indexed column. The record generator supports generating values of *number* and *varchar* data types.

The **Data Structure Information Gatherer** module gathers properties about tables, attributes, constraints, indices and records from **Data Structure Controller** and **Record Controller**, after the user made some changes in **Test Database**. It also needs to send SQL *Select* statement to **Test Database** to gather information about data structures memory usage. If the user changes data structures or data within them, the memory usage will be changed, and this module needs to gather new information about that. Before every gathering of memory usage information, the database statistics need to be refreshed to get the newest information. After all information are gathered, the SQL *Insert*, *Update* or *Delete* statements are sent to **Result Database**.

The **Query Controller** module is used by the user to create a query which will be executed, and which performance will be measured. The user can specify which tables, attributes, functions and conditions will be in the select statement and can choose a specific index to be used. That specification depends on the tables and indices that are already created in **Test Database**. If the user chooses a specific index to be used, the *HINT* clause will be added to the query, to force the index usage. Otherwise, a Query optimizer could decide not to use that index, because it could not be a part of the most efficient execution tree. After the specification of the query is created, the SQL *Select* statement is formed and it is ready to be sent to **Test Database**. To get data from a hard drive, database block buffer cache must be cleared before each query execution. If this wasn't done, sometimes data would be obtained from the hard drive and sometimes it would be obtained from the cache, which would make our test results invalid. After the *Select* statement is sent to **Test Database**, an indicator is sent to **Query Execution Plan Controller and Viewer** to wait for the query execution plan and the query execution time from that database. The user can also create a query and send it to that module to get the execution plan only and not to execute it.

The **Query Execution Plan Controller and Viewer** module can present a query execution plan and query execution time to the user. After the user sent the query to **Test Database**, this module will receive an indicator from the **Query Controller** module, and it will wait for the query execution plan and the query execution time to be received from **Test Database**. This module will present that plan and time to the user and it will send them to **Result Database** in a way of the SQL *Insert* statement. If a query is received from **Query Controller**, the **Query Execution Plan Controller and Viewer** module will send a SQL *Explain Plan* statement to **Test Database** and it will wait for the query execution plan to be received to present it to the user.

The **Result Visualizer** module contains two visualizers: **Query Execution Time Visualizer** and **Memory Visualizer**. These two visualizers use data stored in **Result Database** to present a query execution time or a memory usage of data structures on a chart. The user can choose how the chart will look like and what should be presented on the x and y axes.

If the user chooses to use **Query Execution Time Visualizer**, she or he will need to specify a few parameters. The user needs to specify which queries execution time will be presented, and will it be an average time from more than one execution. Queries execution time will always be presented on the y-axis and the user needs to specify a measurement unit. Different queries could be presented on the x-axis or the same query executed in the different circumstances could be presented. The same query could be executed on different tables, and the number of records or attributes of these tables could be presented on the x-axis. The number of unique values of the selected attributes could also be presented on the x-axis, alongside the cases when some indices are used, or no index is used.

If the user chooses to use **Memory Visualizer**, she or he will need to specify data structures from which memory usage will be gathered. The memory usage of data structures will always be presented on the y-axis and the user needs to specify a measurement unit. Different tables or indices could be presented on the x-axis. The number of records or attributes in the tables or the number of unique values of some attributes could also be presented on the x-axis.

The **Result Visualizer** module is not yet completed, and for now we can only export data from **Result Database** into an excel file to visualize the test results. Some use cases and examples of test results will be presented in the Section 5.

4 Test and Result Databases schema

The repository of ILT is composed of two databases: **Test Database**, storing data for testing purpose, and **Result Database**, storing testing results for analyses.

In this section, we will describe schemas of these databases.

4.1 Test Database

Test Database is used to store the data needed to test and measure execution time of queries and memory usage of data structures. There are predefined data structures that already contain test data. They serve as the example for users to test different queries immediately. The user can create his own use cases by adding new tables and indices and insert new data into them. After that, she or he can create and execute different queries on new data structures and analyze query performance. The **Test Database** schema depends on data structures that the user has created.

4.2 Result Database

Result Database is used for storing query execution time and query execution plans. It also stores properties of data structures, e.g. the name of a table or an index, their memory usage, table attributes and constrains etc. It is the core element needed for the **Result Visualizer** module, which uses its data to present results to the user. In Fig. 2 we present the **Result Database** schema. We will describe every part of its schema in the rest of this section.

The **Table** entity describes tables created in **Test Database**. Every table has a unique name, a number of records, a memory usage and can have more than one attribute. The same table could be selected in multiple queries and used in different execution plans. Many indices could use the same table to index its attributes.

The **Attribute** entity presents any attribute of the tables in **Test Database**. Every attribute has a unique name, an indicator if it is mandatory, a number of different values, a type and it can have more than one constraint. The same attribute could be selected in many queries and used in many indices.

The **Constraint** entity presents constraints that could be created on one or more attributes in **Test Database**. Every constraint has a unique name, a type and if that type is a *Check* constraint, it has an additional condition.

The **Index** entity describes indices created in **Test Database**. Every index has a unique name, a type and a memory usage. It can reference one or more tables and it can index more than one attribute. In cases when a function-based index is created, the string of what is indexed will be stored in the *indexed* attribute. The same index can be required in many queries and it could be used in many execution plans.

Attribute Type, **Constraint Type** and **Index Type** are entities representing commonly used types in database systems. For example, **Attribute Type** can be *Char*, *Varchar*, *Number* etc. **Constraint Type** can be *Primary Key*, *Unique*, *Check* etc. **Index Type** can be *Bitmap*, *B-tree*, *Function-Based Bitmap* etc.

The **Query** entity presents any query that is executed in **Test Database** at least once, and it only

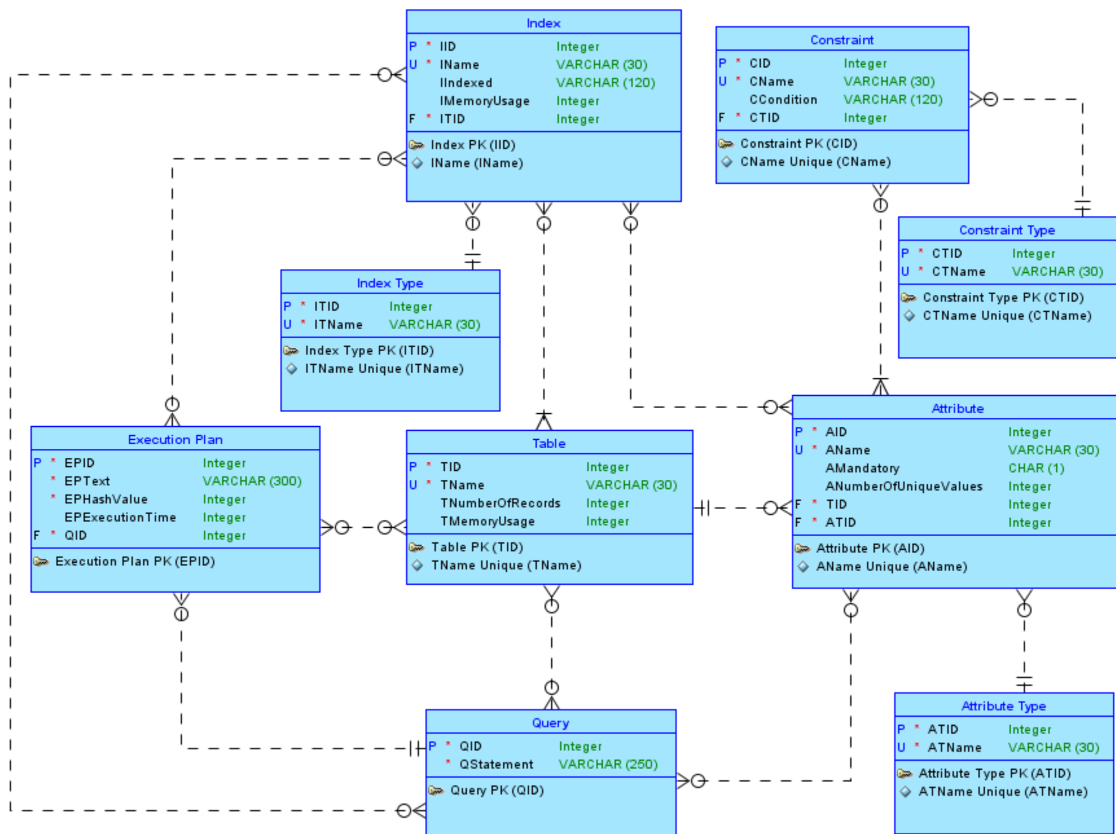

Figure 2. The Result Database schema

has its statement. Every query can select data from many tables and attributes, and it can require the usage of many indices. One query can be executed multiple times and every time the execution plan is stored.

The **Execution Plan** entity presents every execution plan that is created during the execution of any query. It has a text that describes the execution plan, a hash value, a query that represents it and an execution time of that query. Every execution plan can require access to many tables and to many indices.

All these entities, attributes and relations in the schema could be used in many different analyses. For example, we want to present a query execution time using an index in use cases when there are different number of records in tables. To present these use cases, we would need information about that query, a number of records in the tables, a query execution time and execution plans when that index is used. In the next section, we will present use cases when we wanted to measure and analyze a query execution time and a memory usage of indices, depending on the number of records in tables.

5 Use Cases

Students are using our tool to make simple use cases of analyzing a query execution time and a memory usage of indices. Together with our students, we wanted to compare performance provided by bitmap and B-tree indices. Bitmap indices should provide better performance if there are only few different values of the indexed attribute and if bitwise operations or *COUNT*, *SUM* or *AVG* aggregation functions are included in queries. Also, bitmap indices should use less space than B-tree indices if there are only few different values of the indexed attribute. We wanted to practically present these statements to our students by creating experiments that are included in this paper. The use cases presented in this section are not meant to discover a new knowledge. Their purpose is to increase students' knowledge about indexing techniques by demonstrating experiments that are a common knowledge.

Using the tool, students created five tables with twelve attributes. All of them have one attribute reserved for a primary key, one attribute for a testing purpose and ten auxiliary attributes that are needed to increase the table size. All these attributes are of the *Number* type. No constraints were created because that could hinder our test results. For example, creating a primary key constraint could implicitly create a B-tree index on primary key attributes in some DBMSs. Using the Record Generator, students setup parameters that all data would be of the *Number* type and that the test attribute would have 100 different values. They generated one, two, four, eight and sixteen million records in those five tables. In the end, they created the bitmap and B-tree indices on the test attribute for every table. Two queries were created to test the performance using these indices and their memory usage was measured.

In this paper, the Oracle DBMS was chosen for the ILT repository, because we are using it in database courses, and it is one of the most usable DBMSs. Users can choose different DBMSs for the repository, which allows them to compare performance results between the DBMSs of different vendors.

Before every execution of a query, if an index usage was required a *HINT* clause was added. Also, in order to get data from a hard drive the database block buffer cache was cleared.

5.1 Query execution time example

According to (O'Neil & Quass, 1997), using bitmaps with queries that have *AND*, *OR* or *NOT* operations or *COUNT*, *SUM* or *AVG* aggregation functions will be useful, because these operations and functions are fast with bitmaps. To check this statement practically, students created a query with a *COUNT* aggregation function in the *SELECT* statement and with two *OR* operations in the *WHERE* clause:

```
SELECT COUNT(*) FROM table_x WHERE
testcolumn = 40 OR testcolumn = 60
                OR testcolumn = 80;
```

We expected that the bitmap index should provide better performance than the B-tree index because of the *COUNT* aggregation function, *OR* operations and only 100 different values of the indexed attribute. The query was executed ten times on each table and an average query execution time was required. In Table 1., we present an average execution time of the query, that depends on the number of records in the tables, using bitmap and B-tree indices or not using any index.

Table 1. An average execution time of the first query in dependence of the number of records

	Number of records in tables [million]				
	1	**2**	**4**	**8**	**16**
Bitmap index [ms]	22	31	49	57	116
B-tree index [ms]	52	84	97	117	241
No index [ms]	605	1,345	2,399	5,016	9,680

Both bitmap and B-tree indices provided better performance in compare to the test case without using indices. Increasing the number of records, a ratio between an average query execution time of a full table scan and an index access also increases. For example, using 1,000,000 records, the average query execution time using the bitmap index was 27.5 times better than the average query execution time without using indices and for 16,000,000 records it was around 83.4 times

better respectively. That means indices may have even bigger influence when the number of records increases. It is because the *COUNT* function requires an index access only.

The bitmap index provided in average 2.24 times better performance than the B-tree index, which proved our assumption. We can also see that the number of records in the table influence the query execution time using indices, but not as much when the full table scan is done.

Without the *OR* operations, the bitmap index should still provide better performance than the B-tree index, but the ration should be lower than 2.24. To check this statement, students created a query with the *SUM* aggregation function, with no bitwise operations and 100 different values of the indexed attribute:

```
SELECT SUM(testcolumn) FROM table_x
              WHERE testcolumn = 40;
```

The same tables were used like in the last test case. The query was also executed ten times on each table and an average query execution time was required. In Table 2., we present an average execution time of the query, that depends on the number of records in the table, using bitmap and B-tree indices or not using any index.

Table 2. An average execution time of the second query in dependence of the number of records

	Number of records in tables [million]				
	1	**2**	**4**	**8**	**16**
Bitmap index [ms]	26	26	34	36	44
B-tree index [ms]	25	28	38	41	73
No index [ms]	621	1,316	2,401	5,830	10,072

Without using any index, the query execution time was similar to the last test case. We can see that the query execution time using any of these indices is shorter than it was in the last test case. The reason is because in this test case there is no bitwise operations. The bitmap index provided in average 1.19 times better performance than the B-tree index, which proved our assumption that the *OR* operations have a higher impact to the B-tree indices that to the bitmap indices.

Changing table and data properties like the number of table records, the data type of the indexed column and the number of different values of the indexed column could change the query execution time in the number of ways. Also, the query aggregation function, bitwise operations and joins could change the query execution time. We expect that our students make different use cases by combining these table and data properties, queries and indices and test them with the ILT tool, in order to learn more about the impact of these properties.

5.2 Memory usage example

Students measured the memory usage of those Bitmap and B-tree indices created on the five tables and compared them. In Fig. 3., the memory usage of the indices created on tables with different number of records are presented.

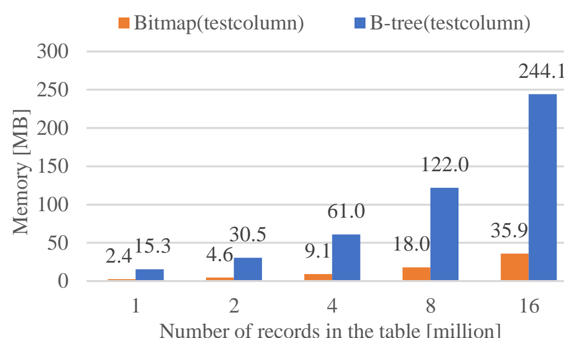

Figure 3. A memory usage of the indices in dependence of the number of records

For each index, we can see nearly linear memory usage growth while the number of records in the tables increases. According to (O'Neil & Quass, 1997), the bitmaps use less space than other indices. Comparing bitmap and B-tree indices in students' test cases, previous statement is confirmed. The bitmap index uses in average 6.67 times less memory than the B-tree index. Creating a bitmap index on the attribute, that doesn't have many different values, will not require too much memory space, comparing to a B-tree index, regardless of the number of records in a table. If there are many different values of the indexed attribute, the results could be different. It is something that we expect of our students to try and learn about through our tool.

The idea is that all these analyses motivate our students do more tests. They can create different use cases changing the index type, the number of records or the number of columns in tables, the number of indexed columns, data type of columns, the number of different values of indexed column, the aggregation function and bitwise operations, and the number of records that are included with the *WHERE* clause. Every change brings new knowledge about the usage of indices and students can learn about advantages and disadvantages of indices in different use cases.

6 Conclusions

In this paper, we presented the ILT architecture and its every module. The tool is created for students to learn about indexing techniques more successfully. Students need to make fast changes on data structures to create

many use cases so they can test performance provided by indices in different circumstances. They can learn in which cases indices are useful and when they are not, and which index type provides better performance in comparison to other types for specific queries and data structures. Providing the possibility of fast changes in interactive environment and performance visualization, we wanted to motivate students to research about indices and easily learn about them with our tool.

We have implemented a prototype of the tool and it still needs improvements. With the Result Visualizer module, we can export data from Result Database and visualize it another way, as we presented in the use cases examples of this paper. In order to completely use our tool only, we must improve that module so it could present charts within the tool. After it is done, we will be able to evaluate ILT with students and probably change and improve other modules. We will try to track students work with the tool and compare that way of learning with a usual way of learning indexing techniques via command prompts or professional tools provided by the DBMS vendors. Our database courses are at the 2nd, 3rd and 4th year of undergraduate studies as well as at the level of master studies. Our students are divided into multiple groups so we can construct an experiment in which we provide certain groups with ILT and other groups with professional tools provided by the DBMS vendors, and compare students' results at the indexing techniques exam. Until now, the tool has been used just with one generation of students and for relevant analysis we will need at least one more generation of students. The first generation of students had mostly positive experience using our tool and their results at the indexing techniques exam were better than the results of the previous generation of students. We will also make a survey with our students about ILT to evaluate their experience using the tool.

We are also planning to add a new module – Materialized View Controller, which will allow students to create materialized views and test performance provided by them. Our students could also compare and analyze a memory usage and an execution time of queries provided by indexing techniques and materialized views in various use cases.

As this tool is designed for anyone who wants to learn more about the usage of indexing techniques, we hope that these new features will motivate them to do more research using ILT.

Acknowledgments

The research in this paper is supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, grant No. III-44010.

References

Agha, M. I. E., Jarghon, A. M., & Abu-Naser, S. S. (2018). SQL Tutor for Novice Students. *International Journal of Academic Information Systems Research (IJAISR)*, vol. 2, no. 2, pp. 1-7.

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *In ACM SIGMOD Record*, vol. 26, no. 1, (pp. 65–74).

Golfarelli, M., & Rizzi, S. (2018). From Star Schemas to Big Data: 20+ Years of Data Warehouse Research. *In A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, vol. 31, (pp. 93–107), S. Flesca, S. Greco, E. Masciari, and D. Saccà, Eds. Cham: Springer International Publishing.

Grillenberger, A., & Brinda, T. (2012). eledSQL: A New Web-based Learning Environment for Teaching Databases and SQL at Secondary School Level. *In Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE '12)* (pp. 101-104). Hamburg, Germany.

Jurgens, M., & Lenz, H.-J. (2001). Tree Based Indices vs. Bitmap Indices: A Performance Study. *International Journal of Cooperative Information Systems*, vol. 10, No. 03, pp. 355-376.

Kenny, C., & Pahl, C. (2009). Intelligent and adaptive tutoring for active learning and training environments. *Interactive Learning Environments*, vol. 17, no. 2, pp. 181–195.

Kleerekoper, A., & Schofield, A. (2018). SQL tester: an online SQL assessment tool and its impact. In Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018) (pp. 87-92). Larnaca, Cyprus

Kleiner, C., Tebbe, C., & Heine, F. (2013). Automated grading and tutoring of SQL statements to improve student learning. *In Proceedings of the 13th Koli Calling International Conference on Computing Education Research - Koli Calling '13* (pp. 161-168). Koli, Finland.

Kung, H.-J., & Tung, H-L. (2010). A web-based tool for teaching data modeling. *Journal of Computing Sciences in Colleges*, 26, 231-237.

O'Neil, P., & Quass, D. (1997). Improved query performance with variant indices. *In ACM SIGMOD international conference on management of data (SIGMOD)* (pp. 38-49). Tucson, USA.

Sadiq, S., Orlowska, M., Sadiq, W., & Lin, J. (2004). SQLator: an online SQL learning workbench. *ACM SIGCSE Bulletin*. 36, 223-227.