Analytical Tool for Oracle SQL Statements

Lukas Jancik, Michal Kvet
University of Zilina
Zilina, Slovak Republic
lukas.jancik.857@gmail.com, michal.kvet@uniza.sk

***Abstract*—Writing statements in Structured Query Language for database querying is one of key activity when learning database systems. Tasks which should be resolved with the stored data using such statements have usually textual character, and evaluation of such statement if it is correct or not typically involves manual check of person responsible for the knowledge provision. The manual check is needed because the correct statement can take various form, which results from the capabilities of the language itself. In this paper, there is presented a solution which performs the evaluation in the automatic way. There are few ways how to compare the statements, the way used is to compare statements itself. This solution deals also with the procession of the input, which is plain text version of statement. This input is considered to come from human who could perform some mistakes in sense of intentional and unintentional mistakes. Unintentional mistakes cover typos and intentional mistakes cover both syntax errors and runtime errors. The software application which was evolved to perform this evaluation is a complex tool, which on one site is highly configurable allowing to modify its behaviour during the evaluation according to user' preference, on the other site it provides all information concerning the execution run to the user.**

I. Introduction

Learning database systems involves not only theoretical aspects, but also practical ones, e.g. practicing of SQL statements creation. If this process is being done on the individual basis with the self-management, then a learning person have a wide range of information they can obtain [1], practice it, and improve themselves. There are also situations, where learning process is covered by some organization, typically a university. In this case, demonstrating gained knowledge is not being left on the learning person solely, but some testing tools are involved. When evaluating answers to testing tasks, it can be simple if we have defined sets of right answers to tasks. Then, the system can automatically evaluate testing tasks and there is no need of human assistance. Set of right answers can be defined typically for tasks which allow as right answers rather small number of the various answers – typically few-word answers, answers in the form of selecting items in the graphical way, or answers which match specific pattern. In case of testing tasks which allow larger number of right answers, there is typically need of human involvement and its manual check, to evaluate given answers. Undoubtedly, testing tasks which require to create a SQL statement based on the given textual description are of this case. SQL statements can be written in different form, and they still can be resolving right one certain task. There can be a huge number of such equal statements, therefore it is very difficult firstly to determine all the statements and secondly supply it as input into the testing system to create set of right testing answers.

There is a clear idea to automate process of evaluation of testing answers just in case of testing tasks which require as the answers the SQL statements. As mentioned, the number of different SQL statements which are still equal in sense of right answer to the one testing task is huge. Moreover, the larger and more complex the SQL statement is, the number is larger. Therefore, the ultimate objective of the system used for evaluation which would solve this issue is not to give 100% certainty that SQL statement which was marked by the system as bad answer to testing task is actually bad. But, if the system will mark SQL statement as right answer to the testing task, then it should be actually right answer – in order to maintain a certain reliability of the system. So, the system should serve primarily as strong supporting tool for persons who will evaluate the statements, allowing them quickly to pass through the answers which were marked by the system as correct, and focus on the problematic aspects of the certain SQL statement more promptly since the output is here from the application. The critical data which the system will rely on is the right answer to testing task in the form of a single SQL statement (typically written by the test creator), not the textual description. This approach is the most suitable, because the test creator can easily provide a right SQL statement and comparing the textual description on one hand to the SQL statement on the other hand, would bring much bigger comparison difficulties than comparing two SQL statements. Also, some other approaches are known how to compare two SQL statements. There is possibility to compare the statements based on the date the DBS returned. This approach was already studied at the Faculty of Management Science and Informatics of the University of Zilina. It showed high time complexity, so its usage in case of multiple testing SQL statements would not be time efficient. The Faculty of Management Science and Informatics covers teaching of database systems on several subjects [2]. The short, in-terrain investigation revealed that no automated approach to process SQL statements (as input into the systems for testing) is available and used. There is a clear window of opportunity to evolve and deploy more specific system for the evaluation. The specificity lies in that teaching of relational database systems is mostly practiced using the Oracle DBS at the faculty, therefore SQL statements are dialect-specific, what means that it will surely need to be handled by the evaluating system.

II. Analysis of System

A. Input data

The input data into the system (for the evaluation) will contain set of pairs, where each pair will consist of the SQL

statement as answer to testing task (mandatory) and the SQL statement as right answer (optional). Both SQL statements will be represented as plain text values – they will not be structured in any way. The right statement is optional, what means that the test creator can and do not have to provide this data. In this case, comparison feature of the system will be unavailable, but some other features of the system will be available to process the testing statement, to reveal potential syntax and runtime errors.

B. Required system features

Undoubtedly, the key feature of the system is comparison of two SQL statements. Apart of this, there are other required features, some of these works on basis of only one SQL statement provided (typically testing statement).

1) Comparison feature: The feature takes two SQL statements and do the comparison. Output of the comparison should not be binary like the statements are equal or not, but there should be detailed view into how the statements are different and which its parts are different.

2) Feature of logs: There should be adjustable logging feature of the system. This feature will monitor all processes during the process of the SQL statements' evaluation. Logs should be gathered in structured way, what will allow a user to filter logs based on some criteria to obtain just the wanted information. The criteria should comprise the level of detail, the type of logs, and the enumerated functional units. In case of the first criteria, the level of detail represents logs of each possible operation, logs aggregated for whole functional unit, and logs aggregated for whole execution. The second criteria should allow to filter the logs based on the type of log, what means by error log, informational log, transformational log, and assessment log. These types are logically concluded based on the required features of the system. The third criteria should allow to filter only the logs which come from the functional unit enumerated. Multiple logs can occur within a single situation – e.g. transformation log and assessment log, e.g. if the system transformed a word with typo and this caused decrease of points in the assessment. Therefore, there is need to bundle multiple logs into some logical unit, to be the situation readable for the end user.

3) Assessment feature: This feature should be able to express the extent of the errors which were detected in the testing SQL statement. Also, it should be clear what is the cause of the minus points given. Therefore, assessment feature and logs feature should be interconnected.

4) Single-Statement evaluation feature: As mentioned in the Input data section, it will be allowed to pass to the system for evaluation the testing SQL statement only. In this system feature, common mistakes should be detected, e.g.:

- Not allowed expression in SELECT part of the SQL statement. Let's take a simple example of this case – having the SELECT statement with the GROUP BY part. The rows are grouped by the columns A and B, in the SELECT part there are listed both A, and B, but also another column – C. The C column is not allowed in this case.
- Unknown column (or ambiguously defined column) in SQL statement – error ORA-00918 (in Oracle DBS). This situation may occur if there is a complex SQL statement with multiple joined tables and at the same time there are referenced its columns in the statement using the simple names. The error occurs when there are additionally referenced columns of the same name which are in multiple tables, in the statement.

C. Other system features

These features cover mostly preprocessing steps.

1) Detection and correction of typos: Typos are common when writing any textual content on keyboard. If the two SQL statements should be compared before this preprocessing step and if the typos would be present here, it would needlessly result in not equality of the statements. It is important to realise that typos can occur anywhere in the SQL statement. They can occur in names of columns, in names of tables, or in key words. The correction should be done based on similar words which come from the set which consists of words, where the originally intended word surely occurs, e.g. word representing a column with detected typo should be corrected based on the similarity measured with the other words (columns) from the same database table. The smaller the set is, and the more different the words within the set are, and the smaller the number of the typos within the single word occurred, then the correction of the word with typo should be the most successful.

2) Procession of aliases of SELECT-part items: These aliases are in most of the cases redundant. They have their reasons why are they used – they simplify expressions so that the expression can be referenced through the alias in different places of the statement. There is even extended possible usage of these aliases, here in the recent version 23c of Oracle Database system – it extends to GROUP BY and HAVING parts [3]. So, the aliases are useful, but in our process of comparison of the two statements, which would have defined various aliases upon the same expressions, it would cause some kind of not equality. Therefore, simple substitution of aliases with their full expressions in every place in the statement and deletion of their definitions would be desired transformation.

III. Analysis of the Application

The system for the evaluation is represented as one standalone software application. This application provides only programmatical access to individual functional units (these are differently divided the system features), not the full service in the form of the GUI and API for the end user.

A. Syntax analysis (parsing) of SQL statements

There is no official way how to obtain or use official Oracle parsing logic in the separate software application

(outside the DBS environment) [4]. As was already mentioned, the input SQL statements are in form of the plain text. In order to process the SQL statements, they will need to have structural form – they need to be parsed firstly. There are two main options how to proceed – new and "built from zero" parsing tool will be created, or existing non-Oracle parsing tool will be used. Syntax of SQL is very complex and creating such parsing tool could be much work and a separate project. Of course, parsing tool in some primitive form could be conducted, but its possibilities in usage would be poor and would be working only in case of the simpler SQL statements. Second choice is to use third-party solution. There are some requirements on the parsing tool – it should be tool, which can be used in the programming language in which the application will be created (Java) (1), the tool should be capable to process specifically the Oracle SQL dialect (2), and it should be well-documented or white-box solution to better understand its implementation and functionality (3). There were found few open-source projects (solutions for free) and one paid tool on the Internet. In case of the paid tool [5], it was not possible to obtain any documentation without buying a license, so this option was not used. From the set of open-source solutions, it was difficult to determine the most powerful one, because no performance-comparison tests were found. The final choice was the solution that appeared as the most profaned and the most discussed in the Internet IT-related communities [6]. This tool is Java SQL parser [7]. The parser converts the textual SQL statement into the tree hierarchy of Java classes, which can be navigated using the Visitor Pattern [8].

B. Design of the application

Application is created in Java programming language, which provide some benefits that can be also practically used. Java programs can run on any device or platform that supports Java Virtual Machine (JVM), making it highly versatile and widely applicable [9]. Java is an object-oriented programming language, which promotes modularity, reusability, and scalability in software development [9]. As it is supposed that development of the application will span long time and maybe the development will even never end, this Java feature is very practical. The long time of the development is concluded from the problem complexity that the application covers.

C. Connection to the database

Connection to the Oracle DBS is important because the application needs to know things like names of columns in particular table, or all table names available. It is need mainly in case of typos in the SQL statements, where the application needs to know the exact names of metadata. When querying database from Java code using OJDBC (Oracle Java Database Connection) service [10], it takes some time to obtain result. When considering real situation that there are multiple testing SQL statements that will need to be processed, it is obvious, that this approach of querying database is not time efficient. Therefore, there should be Object representation of metadata stored in the database – e.g. Java classes representing columns and tables. This way, only few database queries are needed to be performed to obtain all required metadata for purpose of all SQL statements' evaluations.

D. Configuration of the application

The behaviour of the application should be configurable. The user should have the option to pass values, which will modify process of SQL statements' evaluations. There should be option to set the connecting data to the database (1), which cover database URL, username, and password. Another area of configurable values is assessment. User of the application should have option to set extent of minus points for every incorrectness separately (2). This way, user can set strictness of the assessment and can differentiate severity of each incorrectness occurred. For some incorrectness that user consider as not very serious such as typos, they can specify even zero minus points. In many cases, when processing the testing SQL statement, the application assumes that the statement has certain quality to allow preformation of more sophisticated evaluations (or transformations). The assumption can be transformed into the fact right through the specifying it in the configuration (3).

IV. Description of the application

A. Section Typos detection and correction

In this section, typos in column names and table names are considered. The idea is to detect the word with typo(s) and then take the set of the words, where the originally intended and correct word exists. Subsequently, find the correct word and repair the word with typo(s). Basically, the word which is chosen for the repair is the word which have the closest distance to the word with typo(s). There are some well-known metrics which measure distance between two words [11]. In this case, there are words which have been created using the human as a keyboard input. Based on this knowledge, the Damerau-Levenshtein distance is the most suitable metric [11] and is used in the application to compare the similarity of the two words. This metrics treats all common typo's types as single edit – the insertion of character, the deletion of character, the substitution of character, and the transpositions of characters. In the application, firstly the table names and secondly the column names are being repaired. In case of the table names with typos, all the tables from the referenced database are obtained to create the set of the words. Then the mentioned minimum distance approach is used to find the correct table name. In case of columns, the set of words would be too large if all the columns of all the database tables should be taken. At this point, all the table names in the SQL statement are repaired, so the application firstly takes these table names, and the set of words is being created based on all the columns which originate just from these tables.

The process of identifying the columns and tables with the typos inside the SQL statements can be performed by two possible ways using the Java SQL parser. First one is to traverse whole the hierarchy of the Java classes and look for columns and tables and check them whether they have typo(s) or not. The check would compare the word with typo(s) with the corresponding database metadata. The second way how to identify the words with typos is to use parser's feature called DatabaseValidationMetadata [12]. The second approach is used in the application since this feature is available and therefore the traversal within the Java classes' hierarchy is not

needed to be newly implemented. However, the feature is not as complex in sense that it does not allow to repair the detected word with typo(s) directly since it only gives the textual description – not the Java object directly. Within the textual description, there is the name of the column, respectively the name of the table. Therefore, the traversal of the Java classes' hierarchy is needed to be performed, but when column or table is detected, there is no need to perform the comparison with the database metadata, which is still more complex operation then the comparison with the set of detected words with typos. If the column or table which is detected during the traversal is word with typo, then and just then, will proceed to the step where the similarity distances will be calculated between this word and the words from the set. Once the word with typo(s), being it column or table, is obtained during the Java classes' traversal, then the word itself is represented as parser's class representation – it allows to modify it – repair its name.

It is important to mention, that accidental typo creation by the person which performs the input of the SQL statement can result in various situations. Undesirable situations cover such cases as that typo causes transformation of proper column or table to another proper column or table, or when typo caused not proper name but there is not only one but more of the smallest distances to the "correct" words. In the latter case, it is important to aware that there exists other information that can be used in decision of what word should be used for the repair. Let us take one specific case of repair with an ambiguity, which is resolved by the application successfully. SQL statement have in its SELECT part listed 3 columns, one of them was detected to have a typo. The calculation of the distances to words from the set, resulted in two same smallest distances. Therefore, it is unknow how should be the word with the typo(s) repaired. But the SQL statement includes GROUP BY section, where are listed 3 columns, and 2 of them are identical with the 2 columns from SELECT section. The third GROUP BY column is different to the third column from the SELECT section (one with the typo(s)). At the same time, it is the one column from the two columns (which had the smallest distances to the SELECT column with the typo(s)). Based on an assumption that the third GROUP BY column is without a typo and that the statement' creator had complied the GROUP BY rule (which says that only columns in the GROUP BY section can be listed in the SELECT section), then can be inferred that the intended proper name of the column in the SELECT section is just the third one from the GROUP BY section.

B. Section SELECT-part aliases removement

In the SELECT part of the SQL statement, there can be defined aliases of items, which can be simple columns, or more complex expressions. These aliases can be then used in the same statement in different parts (see the Analysis of the system). The application operates with the information whether the SQL statement is targeted to DBS which version is less than 23c, or greater than or equal to 23c. In latter case, process of removement is extended to the mentioned additional parts of the statement. The process of removement of the aliases is much more complex, at least because the aliases can be nested deeply in functions, and there can be nested SELECT statements, from which aliases are taken to the outer level. In case of nested SELECT statements, application uses kind of iterative way, which allows to resolve the aliases removement in full extent independently of number of nesting. Application firstly needs to obtain all nested SELECT statements, it does so by traversing the Java class hierarchy created by the Java SQL Parser. When traversing the tree hierarchy, it proceeds from outermost SELECT statement to innermost statement. Concurrently as the class representations of SELECT statements are traversed, they are being put into stack data structure. It is because when the aliases removement will be proceeding, the aliases need to be removed from the innermost SELECT statement and afterwards proceed towards the outer ones, and in case of using the stack data structure innermost one is located on the top.

When removing the aliases, the application removes indexes too, these indexes can be used in exactly same statements' parts as the aliases and in same parts also with respect to database version as in case of the aliases. SELECT-items aliases removement on level of one SELECT statement (no nested SELECT statements) is relatively a simple task. At this point, SELECT statements are stored separately and chronologically in the stack data structure as they follow in the nesting in the statement. In the one level removement process it is enough to detect all aliases and search them in particular statement' parts, in which they are allowed to be placed. Subsequently, replacement with full expression is being done. Now, when this inner one-level SELECT-items aliases removement is done, outer SELECT needs to be processed. In this situation, it is important to realize that aliases that are now removed in the inner SELECT, can be still used in outer SELECT, in other words, the references are breached. Therefore, removed aliases from inner SELECT need to be saved firstly and removed secondly. Along with aliases also its full expressions need to be saved, because in the outer SELECT the application will substitute the aliases with the full expressions. It is important to note, that not in every case it is possible to perform removement of alias, this concerns nested SELECT statements. If in inner SELECT there is a SELECT-item alias, where the item is not simple column, but more complex expression such as function, then the alias can not be removed in case that the alias is used in the outer SELECT. This is simply because SQL does not allow to pass such expression verbosely from the inner SELECT to the outer SELECT, this is only possible by using the alias.

C. Section of aggregation analysis

In this section, there are treated SQL statements which include GROUP BY part. Once this part is present, there are some rules, that need to be complied in SELECT part, otherwise, the statement will encounter runtime error during execution, and it is therefore an incorrectness in the answer of the testing person. Simple situation would be if there would be only columns in the GROUP BY section and also in the SELECT section. However, GROUP BY item can be complex expression like nested functions with various parameters. Moreover, SELECT item can consists of expression which is

not equal to any of the GROUP BY items but still is allowed and comply with the GROUP BY rule. Such as an expression from the GROUP BY part which is a parameter in a function.

The application takes SELECT items one by one and explores them to evaluate whether it complies with the GROUP BY rule or not. If certain SELECT item complies with GROUP BY rule, then no minus points are given, otherwise they are given. One SELECT item is represented as a single object of multiway tree (data structure) in the application, where the tree' root is whole expression of the SELECT item itself. Children of the tree node are parameters, which can also be functions and can have children too. The GROUP BY part, as was mentioned, can have items which are also expressions such like functions. The multiway tree structure is being traversed in preorder order, which means that firstly is visited current node and after that, children are being visited. If current node being visited is one of the GROUP BY items, being it complex expression or simple column, then children of this node are not being traversed whether they are present or not, because the current node was already found to be present in the GROUP BY section. It was not mentioned directly, but based on the tree structure, it is obvious that one SELECT item can comprise multiple GROUP BY expressions and all of them need to be found the tree structure. If the tree traversal encountered a node which have zero children and the node is column, then it must be in the GROUP BY part, otherwise, minus points are given.

Let us explore situation, where there is given a SELECT statement, in which there is the GORUP BY section with one column and one SELECT item, which is an expression of function, where the parameter is the GROUP BY column. There is no error, and the application will properly inspect the inside of the function and will confirm the parameter as allowed as it is used in the GROUP BY section.

Let us have another situation, which is often encountered among faults of the students. There is a GROUP BY item, which is not a simple column, but function and one of its parameters is a column, or there is some kind of concatenation of textual values and also column is being involved here. The typical situation can look like the item is "SUBSTR(NAME,1,2)" – it means that the grouping of rows is based on the first two characters of the column named "NAME". Then, the fault occurs when on of the SELECT items is the "NAME" and not the whole expression. In this situation, the application will take the column "NAME" and will not find it as a GROUP BY item.

In the process of evaluation by the application, there is one exception, and that is using of aggregation functions. These functions can be used also without GROUP BY section, but once they are used with this section, then the aggregation is being done for the groups of rows, not for whole extent of resulted rows. In case of these functions, also not-GROUP BY items can be listed as their parameters and also the parameters can have each its subtree not occurring in the GROUP BY section – this is said more like in terms of the application logic, but simply said it means that also other columns that are not in the GROUP BY section can occur in such function' parameter.

D. Section of comparison of two SQL Statements

SQL statements are compared in their different parts. One of areas of variations is order of elements, these elements can occur in various parts of the statement, such as SELECT, WHERE, HAVING, etc. There can be different order of these elements in testing statement and in the correct one, and the statements should be treated as equal. However, there are some statements' parts where the order is crucial, e.g. ORDER BY part or functions' parameters. Not only item can be variated in its order, but even its subparts. Particularly, in WHERE part there can be items – logical expressions, which can consist of relational operators. Operands can change their positions vice versa and with change of the relational operator to opposite one, the expression have then the same meaning as the original one. Therefore, the application needs to perform all variations in particular statement' part and then also do variations in items itself. This operation is naturally being done only if the two statements differ in their part. Once there is the difference, then the variations are created. If two statements' parts should be equal, then just one variation should exist in the variations which literally equals to the other statement' part. By other words, the application performs variation steps on one of the SQL statements (do not matter which one) which preserve original meaning of the statement and does the literal comparison with the second statement. If the application finds any statement' part to be not equal to second statement' part, then minus points are given. The application is capable to distinguish some extent of different statement' parts, where it performs the variations and for each such part, not equality is punished with specific amounts of minus points for that part. I already mentioned that there are some statement' parts where order of elements must be fixed in order the original meaning of statement to be preserved. In case of ORDER BY part, the situation is relative complex, because values at specific positions must be same, but their representations can differ, since representation of form of full expression, index, or alias can be used. Moreover, if no direction keyword is being used, then the direction is treated as ascending. Otherwise, keyword "ASC" refers to ascending order and keyword "DESC" refers to descending order. So, the application takes item by item from the ORDER BY part and is looking for its equivalent at the same position in the second statement' ORDER BY part. If the item is index or alias, the application firstly needs to find its full expression. Now, there could be used directly the expression from the first statement, in the second statement. If so, the equality at the first ORDER BY position is confirmed. Otherwise, there can also be used index or alias in the second statement, therefore firstly the transformation to full expression needs to be done and afterwards the final comparison is being done.

V. DEMONSTRATION OF PROCESS EVALUATING SQL STATEMENTS

A. Procession of typos

In the fig. 1 we can see SELECT SQL statement, which have typos in multiple column names. The columns with typos are "_nme" (deletion, "_name"), "surrrrname" (multiple insertion, "surname"), "_description0" (deletion, "_description01"),

"identification_ocde" (transposition, "identification_code"), and "nome" (substitution, "name"). Typos occurred in multiple statement' parts – SELECT, FROM (ON), and GROUP BY. All typos were repaired successfully (fig. 2). With the knowledge of database metadata and tables which are used in the SQL statement, it was enough to repair all typos except the column "_description0". In this situation, two columns with same smallest distances were found to occur in set of words, where the proper name of column exists. These columns are "_description01" and "_description02" and based on usage of one of these two words in the GROUP BY section, the column with typo was repaired to "_description01". This approach is not definitely correct, because typo could occur also in the GROUP BY item "_description01" where the originally intended word could be "description_02" and then the correct word in SELECT part would be this word too. In other words, the application presumes that if the typo occurred in SELECT item and that item is also used in GROUP BY part, then the typo in GROUP BY item does absent.

```
SELECT
    _nme,
    surrrrname,
    _description0
FROM
        personal_data p
    JOIN student_data s
        ON ( p.identification_ocde = s.identification_code )
GROUP BY
    _nome,
    surname,
    _description01;
```

Fig. 1. SQL statement with typos, which is textual input into the application

```
SELECT
    _name,
    surname,
    _description
FROM
        personal_data p
    JOIN student_data s
        ON ( p.identification_code = s.identification_code )
GROUP BY
    _name,
    surname,
    _description;
```

Fig. 2. SQL statement without typos, which is output from the application

B. *Procession of SELECT-items*

In fig. 3 it can be seen SELECT statement with two nested SELECTs. There should be noted, that the innermost SELECT (1) have defined three aliases over three SELECT items ("i", "n", "s"). Also, indexes are used in ORDER BY part at this level. SELECT (2) takes all three aliases from SELECT (1) and use them in SELECT part, WHERE part, and ORDER BY part. SELECT (2) defined new SELECT-items aliases and uses them in ORDER BY part. SELECT (3) takes all three aliases from SELECT (2), defines new aliases over them and uses alias "identification_number" in ORDER BY part.

Fig. 4 depicts output after procession of SELECT-items aliases. When reading the output, it is important to proceed from the innermost nested SELECT to the outermost SELECT. In SELECT (1) indexes in ORDER BY section are replaced with column names and that aliases over these columns disappeared.

```
SELECT 3
    _id  AS identification_number,
    nam  _name,
    surn AS surname
FROM
    (
        2
    SELECT
        i                _id,
        lower(upper(n)) nam,
        upper(s)         surn
    FROM
        (
            1
        SELECT
            identification_number i,
            _name                 n,
            surname               s
        FROM
                personal_data
            JOIN student_data
                USING ( identification_number )
        ORDER BY
            1,
            2,
            3
        )
    WHERE
        n = 'Peter'
    ORDER BY
        1,
        nam DESC,
        s ASC
    )
ORDER BY
    identification_number,
    nam;
```

Fig. 3. SQL statement, which is textual input into the application

In SELECT (2), in ORDER BY section it can be seen, that rows are sorted firstly by "identification_number" ascending, then by "nam" descending, and then by "surname" ascending. Let us break the ORDER BY items down step by step. Where did the "identification_number" come from? In fig. 3 it can be seen that there is index 1 at the corresponding position in ORDER BY section, which refers to SELECT item number 1 – "i" with defined alias "_id". Firstly, index 1 was replaced with column name "i", then alias "_id" was removed. Afterwards, the column "i" was replaced with column "identification_number" (from Fig. 4 – SELECT (1)) in SELECT part and ORDER BY part.

The second ORDER BY item "nam" in fig. 4 comes from the SELECT part where it is firstly defined, so there is no transformation towards the Fig. 3. Alias men is used in the ORDER BY part without any change because it is alias over expression, which is not a simple column. Of course, the alias

could be replaced in the ORDER BY part with its expression, but it is not possible to refer to this expression in outer SELECT (3) – there is alias required, so once the alias is needed, then it is used in the ORDER BY part too. Slight change occurred in the expression itself, over which the alias is defined – it can be seen that on Fig. 3 inside the expression occurs "n" column, while in the Fig. 4 this is transformed to "name". It is because the column "n" does not exist no more after procession of the innermost SELECT (1).

Where did the "surname" as third item in the ORDER BY part in Fig. 4 come from? It came directly from the nested SELECT (1), where it is now transformed (fig. 4), it is also used in "upper" function in SELECT part, where it was also transformed from "s".

```
SELECT 3
    identification_number,
    nam,
    surn
FROM
    (   2
        SELECT
            identification_number,
            lower(upper(_name)) nam,
            upper(surname) surn
        FROM
            (   1
                SELECT
                    identification_number,
                    _name,
                    surname
                FROM
                        personal_data
                    JOIN student_data
                        USING ( identification_number )
                ORDER BY
                    identification_number,
                    _name,
                    surname
            )
        WHERE
            _name = 'Peter'
        ORDER BY
            identification_number,
            nam DESC,
            surname ASC
    )
ORDER BY
    identification_number,
    nam;
```

Fig. 4. SQL statement, which is output from the application.

Both Fig. 5 and Fig. 6 depict SQL statements with no errors. Let us firstly analyse the command in figure number 5. There are 2 columns in GROUP BY section – "name" and "surname". There are 5 items in SELECT part. First item is a

C. *Procession of aggregation*

```
SELECT
    concat(concat(TRIM(upper(substr(name, 1, 2))
                       || '_|_'
                       || lower(substr(name, 3))),
                   '/'),
           surname),
    name,
    surname,
    length(name),
    length(surname)
FROM
    personal_data
GROUP BY
    name,
    surname;
```

Fig. 5. SQL SELECT statement with GROUP BY part showing simple columns in this part

```
SELECT
    COUNT(*),
    MAX(lenght(name))    AS max_length_name,
    MIN(length(name))    AS min_length_name,
    MAX(length(surname)) AS max_length_surname,
    MIN(length(surname)) AS min_length_surname,
    concat(upper(substr((
        CASE
            WHEN length(name) + length(surname) >= 15 THEN
                'YES'
            ELSE
                'NO'
        END
    ),
                        1,
                        1)),
           lower(substr((
        CASE
            WHEN length(name) + length(surname) >= 15 THEN
                'YES'
            ELSE
                'NO'
        END
    ),
                        2)))             AS condition
FROM
    personal_data
GROUP BY (
    CASE
        WHEN length(name) + length(surname) >= 15 THEN
            'YES'
        ELSE
            'NO'
    END
);
```

Fig. 6. More complex statement with complex GROUP BY expression, which is moreover nested as parameter in SELECT item

complex expression, where occurs “name” two times and “surname” occurs one time and both columns are nested more or less deeply in the tree which is represented by this expression (see section Description of application). In case of the first SELECT item, the application behaves like it takes whole the expression and looks for it in GROUP BY section, where it does not occur. Then, the application proceeds with children, which are represented by parameters of the “concat” function and so on, until expression are being found in the GROUP BY section.

In Fig. 6 there is a SQL statement, which operates with data of persons – its names and surnames. The grouping is being done under binary condition – whether sum of lengths of name and surname is greater or equal to 15, or whether the sum is less than 15. Therefore, there is only one GROUP BY item, which is complex expression. Moreover, this complex expression is nested in functions in SELECT item. There can be seen additional SELECT items, one of which lists number of occurrences of records for both sets, other items are aggregate functions, which gives minimal and maximal lengths for both name and surname.

In Fig. 7, SELECT items have different order and partly different naming of aliases than in SELECT items in fig. 8. When looking at WHERE condition in both statements, the conditions are logically the same, but the inner expressions have reversed their operands and relational operators are the opposite ones. Such situation is also in the HAVING part. We can see more complex situation in GROUP BY part, where 3 items are located – ordering ascending by “surname” column, which represents surname of a person, then again ascending by “_name” column, which represents name of a person, and then descending by “identification_number” column, which refer to a personal ID. These two SQL statements are evaluated by the application as same.

D. Comparison of two statements

```
SELECT
    _name                  n,
    identification_number i,
    surname               AS s,
    COUNT(*)
FROM
        personal_data
    JOIN student_data USING ( identification_number )
WHERE
       length(_name) >= 5
    AND length(surname) < 6
GROUP BY
    _name,
    identification_number,
    surname
HAVING
    COUNT(*) > 1
ORDER BY
    s ASC,
    n ASC,
    identification number DESC;
```

Fig. 7: SQL statement which is treated as testing answer

```
SELECT
    _name                  n,
    identification_number i,
    surname                s,
    COUNT(*)
FROM
        personal_data
    JOIN student_data USING ( identification_number )
WHERE
        5 <= length(_name)
    AND 6 > length(surname)
GROUP BY
    _name,
    identification_number,
    surname
HAVING
     1 < COUNT(*)
ORDER BY
    1,
    _name,
    3 DESC;
```

Fig. 8: SQL statement which is treated as right answer to testing task

VI. Conclusion

This paper intends to uncover complex process of evaluation of SQL statements in Oracle dialect. It was shown that third-party (non-Oracle) parsing tool can be used to process SQL statements given as plain texts. The software application, which was described, tends to automatically evaluate SQL statements is sense of their error-free states – both syntax and runtime errors are checked. This application cover only some extent of whole complexity of SQL statements’ comparison and evaluation. Nevertheless, it substitutes human manual evaluation in some extent. The chosen development environment and tools allow to continue in implementation of other features. This paper also intends to describe more specifically some concrete SQL statements and their evaluation to better demonstrate the process of evaluation by the application.

As was mentioned, the application does not cover whole complexity of equivalents when comparing two statements. Therefore, many areas can be resolved by the current application in the future. Especially logical expressions have high variability. Also, variability on level of whole statement should be more investigated. One such example is transformation of SELECT statement with DISTINCT key word, to statement which lacks this key word but the whole statement has been modified in sense that GROUP BY part is added here. No doubt, there are much more such transformations on level of whole statement.

Acknowledgment

It was supported by the Erasmus+ project: Project number: 2022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis (EVERGREEN) funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Slovak Academic Association for International

Cooperation (SAAIC). Neither the European Union nor SAAIC can be held responsible for them.

This paper was also supported by the VEGA 1/0192/24 project - Developing and applying advanced techniques for efficient processing of large-scale data in the intelligent transport systems environment.

REFERENCES

[1] Oracle Database website, Oracle Database Documentation, Web: https://docs.oracle.com/en/database/oracle/oracle-database/.
[2] Web portal of University of Zilina, Curriculum, Web: https://vzdelavanie.uniza.sk/vzdelavanie/plany.php.
[3] Oracle Database website, Database 23c, Web: https://www.oracle.com/database/23c/#group-by-column-alias.
[4] Stack Overflow website, Parser for Oracle SQL, Web: https://stackoverflow.com/questions/5735791/parser-for-oracle-sql.
[5] SQL Parse, Analyzy, Transform, and Format All in One website, Homepage, Web: https://www.sqlparser.com/.
[6] Stack Overflow website, Homepage, Web: https://stackoverflow.com/.
[7] JSQLParser 4.8 documentation website, Homepage, Web: https://jsqlparser.github.io/JSqlParser/.
[8] GitHub page of JSQLParser, Homepage, Web: https://github.com/JSQLParser/JSqlParser?tab=readme-ov-file.
[9] IBM official website, What is Java?, Web: https://www.ibm.com/topics/java.
[10] Oracle Database website, JDBC drivers, Web: https://www.oracle.com/database/technologies/appdev/jdbc.html.
[11] Bealdung website, String Similarity Metrics, Web: https://www.baeldung.com/cs/string-similarity-edit-distance.
[12] GitHub page of JSQLParser, Examples of SQL Validation, Web: https://github.com/JSQLParser/JSqlParser/wiki/Examples-of-SQL-Validation
[13] Karol Matiaško, Michal Kvet, and Marek Kvet, *Databázové systémy – 1.diel*. Žilina. EDIS-vydavateľské centrum ŽU, 2018.