Numeric Fields in Database Development: From Optimal Design to Secure Coding—SQL Attacks Protection Method[⋆]

Yaroslava Momryk[1,†] and Dmytro Sabodashko[1,*,†]

[1] Lviv Polytechnic National University, 12 Stepan Bandera str., 79013 Lviv, Ukraine

Abstract

The paper aims to develop an optimal approach to database (DB) development—from database design, including the selection of field types, to developing protective techniques for working with databases, particularly to safeguard against SQL attacks. The choice of numeric fields for constructing primary and foreign key identifiers is justified from the perspective of optimizing database development steps and ensuring their protection during use. An analysis was conducted to implement recommendations for working with DBMSs, emphasizing mandatory verification of input fields on the server side. This issue is addressed in terms of the correct selection of field types during database structure development to optimize the resolution of such issues in subsequent development steps, particularly in establishing necessary validation criteria and developing protective methodologies for client-server applications. The developed structure and methodology for data validation reduce the need for additional frameworks or expensive tools when defending against attacks such as SQL injection and ensuring personal data protection. The proposed strategy covers the construction of interaction keys based on the method of adding a fractional part, along with its algorithm and model for client-server applications, including consideration of the specifics of software implementation. In particular, aspects of the programmatic implementation of the method of adding the fractional part for identifiers represented by large numbers are considered. Key construction methods are proposed to implement this method, including key construction and an algorithm for applying these keys to improve the use of the length truncation method. Criteria and examples of software implementation for transferring identifiers from the client to the server are developed, alongside an algorithm for hiding key identifiers in record processing forms as part of the protection model against SQL attacks. Developers are advised to use numeric field types for creating primary and foreign keys to ensure safe and optimal database and application development. A strategy and methodology for protection in database and application development are outlined to implement the security-as-code approach in alignment with modern development standards.

Keywords

database design, database optimization, data protection, key field, numeric database field type, security as code, SQL implementation, web application security, HTTP request

1. Introduction

Since data threats are becoming so relevant that the authors [1] use the term “cyber-war”, it is suggested to comprehensively solve this issue, taking into account the internal threats associated with the imperfect development of the structure and software, which, together with the mistakes of the staff lead to integrity and data loss, and aspects of attacks on databases over the Internet. It is important to adhere to the goal of optimizing database work. Considering database optimization techniques, the authors [2] recommend effective database schema design as one of the steps to achieve performance: “Design your database schema with performance in mind. Optimize data types, use appropriate constraints, and minimize redundant relationships. A well-designed schema can significantly affect query performance.” In general, possible optimization methods [3] (directly the optimization of the database and the database as a whole; optimization of the interaction of the program and MS SQL Server; optimization of queries) depend on the correctness of this step, which is illustrated in the presented material.

[⋆] CPITS 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2025, Kyiv, Ukraine
[*] Corresponding author.
[†] These authors contributed equally.

yaroslava.b.momryk@lpnu.ua (Y. Momryk); sabodashko.dv@gmail.com (D. Sabodashko)
0009-0001-4114-0347 (Y. Momryk); 0000-0003-1675-0976 (D. Sabodashko)

According to [4], the correct choice (professional approach) to the design stage is the way to ensure data integrity and security and provide data protection from internal threats at subsequent stages of work, and it is about databases that the application of the secure code methodology begins at the database design stage.

Problem formulation

Despite existing protection strategies, SQL injections are effective attack vectors [5–9]. Therefore, further research and development of new techniques and recommendations are needed to prevent these attacks effectively. When it comes to databases, the primary threat is SQL injection attacks, which, depending on the type of database used and the implementation conditions, can allow an attacker to execute arbitrary queries against the database [10]. When SQL attacks are successful, attackers can [5–9]:

- Log in to an application or website without a password.
- Access confidential data, and extract and delete stored data from protected databases.
- Opening the door for further attacks.

Therefore, it is time to consider the question of how to properly develop the structure of the key field database for relational database tables so that it meets the possibility of optimally solving the above objectives and consider the method of protection, in particular against SQL attacks, using this structure at the subsequent stages of development and operation.

Recent research and publications analysis

A wide range of literature covers issues of database security and optimization—from reviews of the types of most common database threats [8] and approaches to database optimization and interaction with them [2] to attack vectors and countermeasures. In particular, methods of attacks and countermeasures against SQL attacks were analyzed [11–19].

The review [6] globally covers attack vectors and approaches to countermeasures, in particular, it emphasizes that SQL attacks can also directed at the disclosure of identifiers for the leakage of personal data; among the attack vectors for data theft are SQL injection and in particular with the use of scripts [6] "in many cases, the attacks are trivially automated for attackers by downloadable scripts" that is why the research focused on the aspect of vector script attacks.

In [9], there is a function that performs validation according to the markers—the authors are based on the necessity of data checks during the interaction of the database with the client and the server.

Researchers [20] reveal another direction of threats—"defines a class of SQL-injection attacks that are based on injecting identifiers, such as table and column names, into SQL statements".

The analysis of the literature gives grounds to conclude that the majority of attacks today are exploitation of vulnerabilities of the application itself—"security has become one of the main challenges in recent years because most of the web applications have suffered from vulnerabilities that have made them attractive targets of security attacks" [7, 21].

Based on the above, an attack of the SQL injection type may be possible due to incorrect processing of input data used in SQL queries; therefore, in this study, primary attention is paid to the validation of data received by the server from the client—due to modern threats, it is practically forbidden to take data from the request array into a query without checking [16, 17].

In the analyzed literature, as a rule, for protection, we see recommendations for using protective frameworks, encryption, access priorities [6], and other methods that allow application programmers to avoid writing code for deep checking when developing application programs for interaction with the database. Among the proposed methods are frameworks that are effective for protection in the case of implementing tasks that require special protection—in particular, labor-intensive methods, such as [22] system that automatically transforms fragments of application logic into SQL queries *f*, *j* or *a* framework for working with login fields [23] and others. Some authors suggest using advanced or special technologies such as ORM [10] or blockchain [24] to protect data.

Implementations of the simplest steps, such as transitions to a page with a selected product, for example, without cumbersome and often expensive and resource-intensive developments (frameworks) in the direction of secure code are lacking, and the issue of the structure and types of fields in this aspect is not considered at all.

But SQl-injections largely refer to attacks on URLs and HTTP requests, where there are field keys and requests to select or modify the desired information based on the identification keys of the selected information transmitted by the user program.

Therefore, the main emphasis of this study is devoted to the stage of building the structure in terms of selecting the types of these fields and the construction of the methodology presented to confirm the convenience and optimally easy implementation of protection on numeric field types.

The authors [8] consider internal and external risks to databases, calling along with the risks from SQL injection and internal threats "including programmatic errors that programmers make while building networks, or designing different applications."

This is the basis for finding a solution that prevents the creation of gaps through which attacks are carried out. In [4] recommended building data protection based on a professional approach, starting and proceeding from the optimal construction of the database structure, in particular the primary and foreign key fields—this study is about the correct selection of the table structure and key fields—continuing the above, we justify that this is the first step in building optimal protection against both internal errors and external attacks. Considering the requirements of professional implementation and the characteristics of the attacks themselves, we conclude that such a comprehensive approach provides optimal protection in modern conditions [25, 26].

Software certification expert Both experts such as M. Jetelka [27] and researchers [28] write about the high importance of using secure coding in software development, which is reflected in the updated information security standards. As a response to the above-mentioned challenges of modernity and the corresponding update of standards, the proposed material focuses not on creating another security framework but on developing an optimal approach to developing a database structure and developing a data validation methodology for a protection methodology that can be used as secure code and is simultaneously a method for optimizing work with databases.

The purpose of the paper

The purpose of the research is to consider the applied area of relational database design and software for working with them from the point of view of key components of design and development that affect the optimization of work and information security, to substantiate recommendations on the choice of numeric types for constructing key fields in databases to optimize work and counteract SQL injections, to develop a model, practical algorithms and strategies, and to build a methodology for countering these attacks. Tasks:

1. Analysis of the feasibility of constructing a database structure based on the concept of basic identifiers using numeric fields. To substantiate and show ways of using them from the point of view of optimization and protection against internal and external threats.
2. Formulation of optimal validation criteria—a simplified methodology for countering attacks, which can be used both without special-purpose frameworks and as part of them.
3. Development of a method for protecting against distortion of identifier values when transmitting identifiers from the client to the server using the principle of adding the fractional part and software with an example and improving its implementation, including for identifiers of large length.
4. Development of a method of use and an algorithm for the truncation method when the length of identifiers is exceeded.

5. Development of a model for improving the method of hiding key identifiers when editing data in web forms.

2. Justification of the choice of the numeric type of identifiers for database protection

2.1. Justification of approaches to structure design

Let's consider the arguments for working with databases, based on which we can recommend developers use numeric field types for building primary and foreign keys:

1. **For key formation.** With a number, there is less chance of errors, including errors when building an external connection—if this is a key field—it is either generated by a procedure that uses something like sequence or is set as auto-incremental in the structure, or there is a procedure that generates a cipher—that is, this field is not typed by hand—accordingly, there are no errors from the client to the server due to a random change in the font language and encoding.
2. **For indexing (one of the most important optimization methods).** If you index—because indexes are created for the primary key and foreign keys at the DBMS level—such an index (including clustered) will take up less space. Thus, we will avoid creating [3] indexes, which use more costs for their maintenance than they bring benefits.
3. **For search speed** (including using index fields). For the DBMS to compare two strings, it is necessary to compare two strings character by character. The search for numeric identifiers is faster due to the very technology of performing such an operation for numbers.
4. **To reduce the probability of errors**. When selecting data by join, when connecting using character fields, there is a greater probability of errors due to inattention (if it is assumed that such fields are corrected manually), a change in encoding, or language (some letters such as 'a' look the same in different languages, but have different codes), therefore the probability of integrity violation due to internal errors is much lower for numeric fields, so they should be set as foreign key fields.
5. **To reduce bulkiness**. To avoid partitioning [2]—which is recommended as an optimization method. Tables that will have numbers about other tables will be less bulky in size.
6. **To build not bulky but effective protection methods**. This task is accomplished with less effort and less resource burden than when working with symbolic key fields due to the speed of operations with numbers and the speed of comparing numbers if there is a need to process such an identifier through symbolic variables—a number is easier to identify and detect an SQL injection because it consists of digits.

If the identifiers of primary and foreign keys, which are usually used to navigate to the desired page or select certain information, are built using an integer, then to prevent attacks, it is easy to cut off most threats in the sense of secure coding. This is especially relevant if we work with a database using vulnerable technologies (technologies other than ORM)—that is why the examples are built on such technology (php+MySql).

Next, it is demonstrated that GET links on the web with the transfer of numerical parameters are much easier to control and protect against SQL attacks than symbolic ones (POST will work similarly). In the examples, the emphasis is not on the construction of special methods but on development techniques within the framework of implementing the "safe code" principle; the logic is shown in Fig. 1.

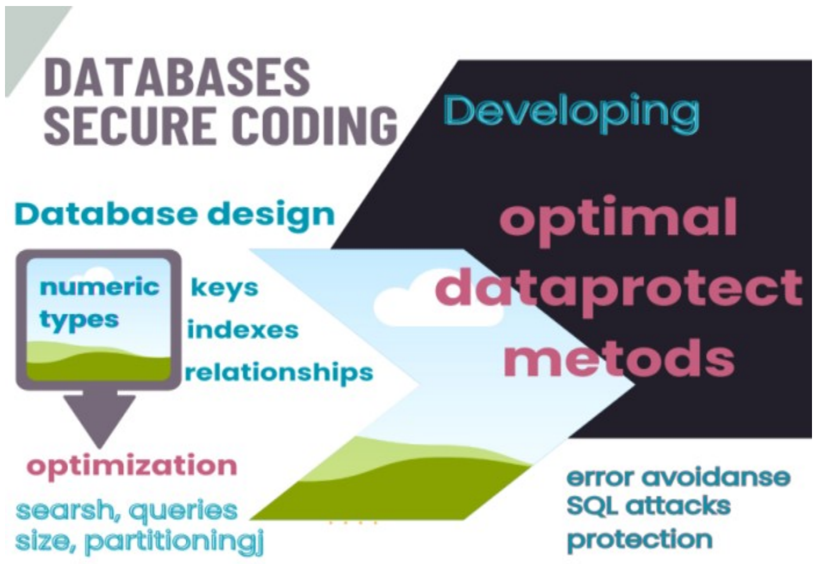


Figure 1: Using numeric field types in database development

2.2. Construction of the principles the principles of the method for protecting numeric types of identifiers and implementation examples

Based on the methods of SQL attacks described in [11, 14, 15, 18, 19] that can be applied to identifiers and key fields, we will consider the protection strategy and check whether it protects the following methods of attacks:

- SQL injection attacks
- Attacks using UNION
- Attacks using the SQL query separator
- Attacks adding numeric data to identifiers
- Application-level resource exhaustion attacks [9].

As for the last point, if we are talking about a professional implementation that will protect against internal threats due to employee inattention and external ones—due to a deliberate application-level attack aimed at exhausting database server resources, in this aspect, attention should be paid to the transmission of a space, which gives the effect of emulating the load on the server. Therefore, if the programmer passes empty fields into the query—for example, when a space is put in the search field, or an empty identifier is passed in the get search parameter, then he becomes a conductor of that attack, so it is necessary to check such moments immediately and not form an unnecessary request to the server. In the examples below, this moment is checked before forming queries in both numeric and character variables passed to the server.

2.3. Criteria for the simplest defensive validation and examples of their application to intercept SQL attacks

Let us consider the advantages of using numeric identifiers when building protection methods using an example where the client form sends an identifier to the server from the REQUEST array (consists of variables passed by the client via $_GET, $_POST, $_COOKIE)—as in Fig. 2

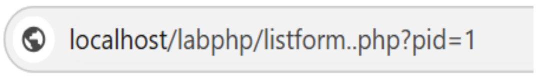

localhost/labphp/listform..php?pid=1

Figure. 2: Passing identifiers via page URL

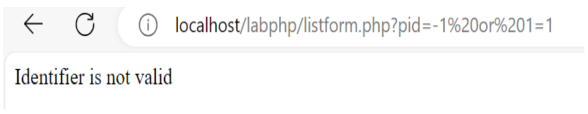
With the help of numerical identifiers, the above attacks can be easily blocked if the following simple verification criteria are used: if the identifier is non-empty and if it is a number.

A sample of such a check is given in the code block below—such a check excludes the possibility of adding malicious code. Moreover, this validation method will work for different DBMSs—regardless of whether they convert a character parameter to a numeric one in case of a type mismatch.

```
if(isset($_REQUEST['identifier']) && trim($_REQUEST['identifier']) != ''&&
is_numeric($_REQUEST[''])){
    $servidv=intval(trim($_REQUEST['identifier']));
```

It has been tested that such validation protects against the following threats:

1. **Attack based on the principle of SQL embedding injection**. If the program that processes the request uses the validation below, then the attacker will not achieve anything by replacing identifier = 2 with identifier = -1 OR 1 = 1 (Fig. 3)

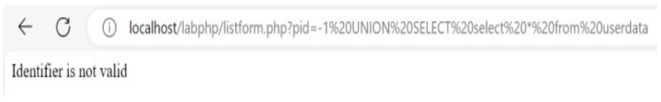


localhost/labphp/listform.php?pid=-1%20or%201=1

Identifier is not valid

Figure 3: Intercepting an SQL injection attack

2. **Attack using UNION.** If a hacker passes something like -1 UNION SELECT select * from userdata to id, the query will also fail because it contains characters—validation will reject it (Fig. 4).

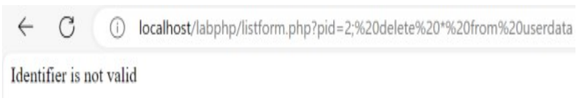


localhost/labphp/listform.php?pid=-1%20UNION%20SELECT%20select%20*%20from%20userdata

Identifier is not valid

Figure 4: Intercepting an attack using UNION

3. **Attack using the SQL query separator**. The ";" (semicolon) character is used to separate commands in SQL. By introducing this character into a query, an attacker can execute multiple commands in one query, but not all SQL dialects support this feature. For example, if a hacker tries to add a delete query—identifier=2; delete * from userdata—it will also not work (Fig. 5).

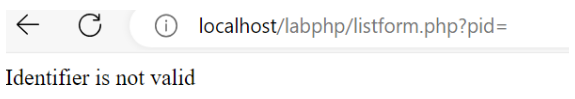


localhost/labphp/listform.php?pid=2;%20delete%20*%20from%20userdata

Identifier is not valid

Figure 5: Intercepting an attack using the SQL query separator

4. **Attack based on trying to emulate the load—entering empty fields and spaces.** In all cases, validation works and the program issues a message as in Fig. 6.



localhost/labphp/listform.php?pid=

Identifier is not valid

Figure 6: Message about interception of attack when validating a field composed of spaces

For symbolic values, it is better to use placeholders—parameterized queries [29]. However, prepared parameterized queries slow down the server and create more load—therefore, numeric identifiers provide optimization when programming works with databases. This method allows you to work only with valid identifiers. If the identifier does not pass validation, the program operation is interrupted.

3. Methodology based on the method of adding fractional parts to form interaction keys

3.1. Applying the method to neutralize the attack of appending numerical data to identifiers

Regarding the attack of appending numerical data to identifiers of identifiers, start transferring identifiers via HTTP GET in the logic of PEN testing and consider that if SQL instructions convey the possibility of appending data to identifiers ([11] suggests such methods of use all up to the maximum allowed duration)—one should consider the case when the validation developed above is not enough. Such a situation will arise if a hacker tries to redirect another record—append numbers to the formed URL path with the passed parameter to provoke an integrity violation and corrupt the data, for example, when editing the record. For example, the user program selected a record with identifier 5, and the hacker appended more numbers—the identifier became 55 and participates in the work with completely different data, or the identifier will not be found in the list of existing ones. To avoid such distortions, below we offer a method as the development of the simplest method of checking numerical identifiers for parameter protection by fixing and controlling the length of the parameter.

To protect against an attack, we need to ensure that we get exactly the identifier the user chose. To control the length of the parameter, we can change the numerical identifier to transfer it to the program and convert it to the float type, where the fractional part will contain the character length of the identifier—the method of adding fractional parts. If we are talking about a database of average size, the identifier can be sent to the server in the form—“id value”.”length value” using the following code [30]:

```
$ilen=strlen((string)(trim($row['id_person'])));
$st=strlen((string)(trim($ilen)));
$llen=(int)$row['id_person']+$ilen/10**$st;
print "<tr><td>{$row["name"]}</td><td>{$row["pfone"]}</td><td><a href=\"listform.php?identifier=$llen\">Edit</a></td></tr>";
```

Then, the program that processes the data on the server can check the length and either strip off the extra characters or abort execution – in the example below – a warning is displayed.

```
if(isset($_REQUEST['identifier']) && trim($_REQUEST['identifier']) != ''&& is_numeric($_REQUEST['identifier']))
   {
    $idv=floor(trim($_REQUEST['identifier']));
    $len=round(trim($_REQUEST['identifier'])-$idv,2);
    $llen=(int)(substr((string)($len),2,2));
    if(strlen((string)$idv)!= $llen)
     {
      print "LEN IDENTIFIER IS NOT VALID";
     }
```

It is worth noting that the example is made for key numbers whose length is described by a number that takes up no more than two characters—there are no larger identifiers in this aspect of working with databases. If transferring this data, the remaining characters are discarded to protect against adding characters to the end of the identifier of the transferred value.

An illustration of the operation with the correct identifier length (=2)—in URL displayed 18.2, where 2 is the identifier length is shown in Fig. 7.

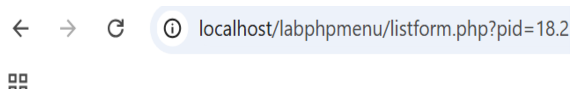
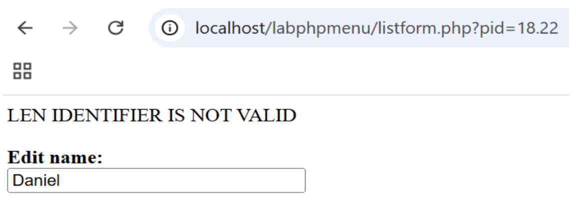


Figure 7: Program operation using the method of adding the fractional part with the correct identifier length (=2)

If the attack vector is constructed in such a way that digits are added to the end of the already formed fractional number, then this will not harm the work of the samples—the identifiers will be correct because digits will be added to the fractional part—and it is automatically discarded in this method—therefore, by default, only a warning is issued, and the work is not interrupted (Fig. 8).



LEN IDENTIFIER IS NOT VALID

Edit name:
Daniel

Figure 8: Neutralization of hacker attack of adding numbers using the method of adding the fractional part

The above implementation is focused on working with medium-sized numbers—it is not suitable for processing large numbers (if the length of the identifier in characters is more than 12, distortions are possible). For a universal implementation of the method, it is necessary to take into account the specifics and method of storing fractional numbers—it is advisable to transfer them as a string and, upon receipt on the server, perform conversions for validation that will prevent data distortion in the fractional part due to the specifics of processing fractional numbers. A code like the one shown below will work correctly.

Code for transferring the identifier:

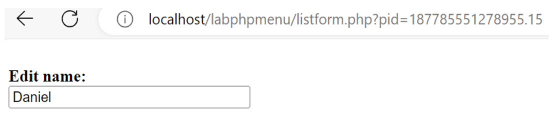
```
    print "<form method=post action='listform.php'>";

    $ilen=strlen((string)(trim($row['id_person'])));
    $llen=$row['id_person'].".".$ilen;
     print($llen);
     print "<tr><td>{$row["name"]}</td><td>{$row["pfone"]}</td><td><a href=\"listform.php?
identifier=$llen\">Edit</a></td></tr>";
```

Example of a validation code for a transmitted identifier:

```
    if(isset($_REQUEST['identifier']) && trim($_REQUEST['identifier']) != ''&& is_numeric((int)
$_REQUEST['identifier'])){
      $idv= intval (trim($_REQUEST['identifier']));
      $pos=strpos(trim($_REQUEST['identifier']),".")+1;
    $llen=substr(trim($_REQUEST['identifier']),$pos,2);
      $idvl=strlen((string)$idv);
      if ($idvl!= $llen)
       {
        print "LEN IDENTIFIER IS NOT VALID";}
```

Such a software implementation of the method will work correctly with large numbers (Fig. 9), but it is still necessary to take into account the peculiarities of the PHP programming language—for processing very large numbers of the bigint type, conversion to int will not work correctly—the number will be truncated.



localhost/labphpmenu/listform.php?pid=187785551278955.15

Edit name:

Daniel

Figure 9: Implementation of the method for large numbers

These moments no longer apply to the method and are solved by standard methods for the programming language—you can connect libraries for processing large numbers of the bigint type or, as an alternative—process as a string and use a regular expression to validate for the presence of only digits.

To illustrate the operation of the method, a warning is programmed, but you can design, according to the needs of the project, the creation of interrupts or the processing of such situations depending on whether the identifier is longer or shorter than the number specified in the fractional part. But if it is possible to isolate the infection from the identifier, it is better to log—collect information about such failures so that the administrator can analyze the attack method and choose a strategy as described [16]—in terms of the usefulness of using logging. You can choose the method of software processing according to the values of key identifiers.

3.2. Improving the length truncation method

The proposed approach can be applied to improve the truncation method used in counteracting SQL injection—when unnecessary characters are cut off in the parameter to prevent overflow of variables. For example, we can truncate an identifier—so that its length has as many characters as is fixed in its fractional part. Here, a checksum will come in handy. Moreover, it is advisable to build a checksum not by simply adding digits—because the identifier can be changed by rearranging the same digits as in the original, but taking into account the index (entry number) of each digit in the number. For example, we can build the sum of digits multiplied by their ordinal number in the identifier (1). Then, when changing the order of the identifier numbers, the checksum (Skident) will also change.

$$Skident=\sum_{i=1}^{n} x\,i*i \tag{1}$$

where x is the current digit of the number; *i* is the ordinal number of occurrences of this digit; *n* is *the* count of digits of the number.

For example, for Id= 32578, the checksum will be calculated as follows: S=3*1+2*2+5*3+7*4+8*5.

Then the already transferred identifier will be once: ddddddddd.dddddd.dd where before the first dot is the identifier, after the first dot—its checksum, after the second dot—the length, while d is any digit.

Then, the validation algorithm will look like this:

1. Select the first two fragments, separated by "." from the tape, and check if they are numbers.
2. Check the correspondence of the first two fragments—the identifier and its checksum.
3. If there is no match and the identifier is longer than the length specified in the third number, trim to the specified number (cut the length to 2 characters if the number passed is longer), and recheck the result with the checksum.
4. If the corrections helped, we can continue working only by reporting the failure; otherwise, we can generate an interrupt.

So, this method improves the performance of protection against SQL injection and the threat of length overflow—as a result of validation, we can continue working even if there was an attack attempt—in all cases when malicious code is added to the identifier, which is the most common way of attacks on key fields, the added code is simply discarded, and the server can get the correct identifier from the integer part of the number. The same checksum can be passed in the fractional part of the method built above for passing identifiers from forms. To perform such calculations, it is advisable to create a function.

3.3. Record editing method with hiding the key identifier field

Above, we are talking about primary SQL infections when transferring the record identifier using the get method or through hidden post fields—they are based on adding malicious code and not on completely replacing the transferred value—these are methods of attack that were tested according to those described in the analyzed literature. However, attackers also attack HTML headers and cookies [6]. If the identifier is stored, for example, in cookies, then to detect whether this data has been maliciously replaced, it is also convenient to build a protection method in combination with the method of adding the fractional part. For this, we can generate cookies with the value of the identifier checksum. The values of strategically important key fields should not be made visible either through passive observation attacks to prevent vector attacks in the case of auto-incrementing key values or to prevent access to personal data or data such as medical records or bank transactions—we are not talking only about theft, but also about the possibility of spoofing [31] and data corruption—these are threats that are named as the first two threats in SQL attacks also in [9].

Therefore, at the stage of implementing a regular application that is not protected by special methods, it is worth taking care of hiding key identifiers. Here, you can combine the methods discussed above. To illustrate this approach, consider the response of a record on a web page. When programming this operation, as a rule, the value of the identifier is hidden from the user, passing it through a hidden field as shown below (but its value can be seen by looking at the page code, parsing will also allow you to get the value):

```
<input type='hidden' name='identifier' value='1257' size=20>
```

This method of forming edit forms can be improved and built as follows:

- Store the identifier value in cookies in the format provided by the method of adding the fractional part.
- Transfer only its checksum to the edit form and perform a check—compare this data on the server before forming the request.

If an attacker tries to spoof a cookie, this method can also prevent it from being used by malicious scripts—the identifier in the cookie does not match the real one in the database and requires confirmation from the form that interacts with it.

It is advisable to log warnings about attack attempts and analyze them to choose a defense strategy.

The general model of the developed methodology is shown in Fig. 10.

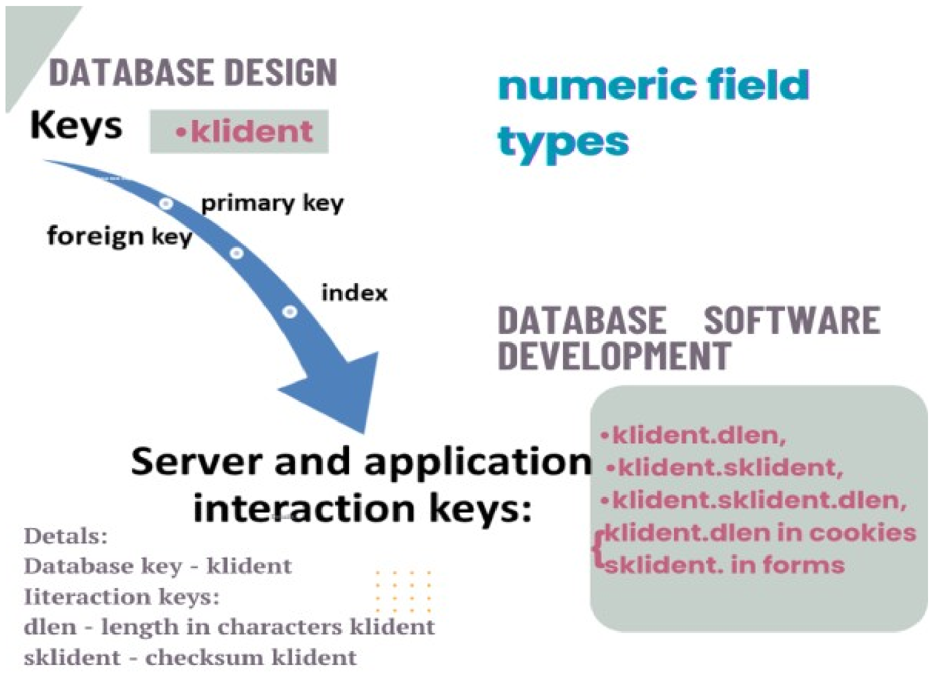


Figure 10: Validation model with the addition of a fractional part

Conclusions

The above analysis states that using numeric-type keys for primary key fields and external data when designing database tables makes it possible to facilitate data protection, particularly against SQL injections, and optimize work with the database. Based on this structure, a criteria validation and a methodology for protecting against substitution or distortion of key identifiers during transmission from the client to the server have been built to implement secure development. The method is flexible: in the fractional part, other criteria for checking validity can be applied.

In the future, it is planned to investigate the possibility of applying the presented approaches to uid and the format of their storage in databases and to improve the methodology for recognizing the method of SQL attack and providing the ability to make a decision based on the logged statistics of recorded failures: choosing a method of counteraction—for example, whether to truncate the identifier before forming a query to the desired value or whether to form an interrupt.

This material embodies the application of the “security as code” approach, starting from the database design stage—when establishing the table structure and up to the stage of interaction with software applications for working with databases. The method can be connected to protective frameworks, implemented in such a way as to be able to choose the mode of processing the errors recorded by it and to connect an intelligent component that suggests the most appropriate method.

Acknowledgments

The authors would like to thank the Ph.D. Y. Yashchuk, Head of the Department of Applied Mathematics, Ivan Franko National University of Lviv, Ukraine, provided valuable suggestions and advice on the presentation of the material.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication’s content.

References

[1] O. Haiduk, V. Zverev, Analysis of cyber threats in the context of rapid development of information technology, Cybersecur. 3(23) (2024) 225–236. doi:10.28925/2663-4023.2024.23.225236

[2] D. Foo, 11 database optimization techniques, 2023. URL: https://danielfoo.medium.com/11-database-optimization-techniques-97fdbed1b627

[3] I. Chyhira, V. Levitskyi, A. Mykytyshyn, Stages of database optimization, in: 4th All-Ukrainian Scientific and Technical Conference Theoretical and Applied Aspects of Radio Engineering, Instrumentation and Computer Technologies, Ternopil National Technical University named after Ivan Pulyuy, 2019, 256–257.

[4] Y. Momryk, Y. Yashchuk, R. Tuchapskyi, A professional approach as a method of protecting information at the stages of development of relational databases and software for working with them, Cybersecurity 3(23) (2024) 42–55. doi:10.28925/2663-4023.2024.23.4255

[5] S. A. Faker, A. M. Mohamed, H. Dachlan, A systematic literature review on SQL injection attacks techniques and common exploited vulnerabilities, Int. J. Comput. Eng. Inf. Technol. 9(12) (2017) 284–291.

[6] F. Ullah, et al., Data exfiltration: A review of external attack vectors and countermeasures, J. Netw. Comput. Appl. 101 (2018) 18–54. doi:10.1016/j.jnca.2017.10.016

[7] A. S. B. Shibghatullah, et al., A comparative analysis and performance evaluation of web application protection techniques against injection attacks, Int. J. Mob. Commun. 18(1) (2020). doi:10.1504/ijmc.2020.10019530

[8] A. Mousa, M. Karabatak, T. Mustafa, Database Security Threats and Challenges, in: 2020 8th International Symposium on Digital Forensics and Security (ISDFS), IEEE, 2020. doi:10.1109/isdfs49300.2020.9116436

[9] R. Johari, P. Sharma, A survey on web application vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection, in: 2012 International Conference on Communication Systems and Network Technologies (CSNT), IEEE, 2012. doi:10.1109/csnt.2012.104

[10] P. Petriv, I. Opirskyy, N. Mazur, Modern technologies of decentralized databases, authentication, and authorization methods, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3826, 2024, 60–71.

[11] W. G. J. Halfond, A. Orso, A classification of SQL injection attacks and countermeasures, in: ISSSE, 2006.

[12] Veracode, Secure coding best practices handbook. URL: https://info.veracode.com/secure-coding-best-practices-hand-book-guide-resource.html

[13] Z. S. Alwan, M. F. Younis, Detection and prevention of SQL injection attack: A survey, Int. J. Comput. Sci. Mobile Comput. 6(8) (2017) 5–17.

[14] C. Anley. Advanced SQL injection in SQL server applications, 2002.

[15] Web Security Academy, What is SQL injection? Tutorial & examples. URL: https://portswigger.net/web-security/sql-injection

[16] PHP: SQL injection—Manual. URL: https://www.php.net/manual/en/security.database.sql-injection.php

[17] SQL injectionSQL Server. URL: https://learn.microsoft.com/en-us/sql/ relational-databases/ security/sql-injection?view=sql-server-ver16

[18] A. Paul, V. Sharma, O. Olukoya, SQL injection attack: Detection, prioritization & prevention, J. Inf. Secur. Appl. 85 (2024) 103871. doi:10.1016/j.jisa.2024.103871

[19] W. Deng, P. Yan, Security risk and preventive measures of multimedia database system under remote control of network robot, J. Robot. 2023 (2023) 1–8. doi:10.1155/2023/9276208

[20] C. Cetin, D. Goldgof, J. Ligatti, SQL-identifier injection attacks, in: 2019 IEEE Conference on Communications and Network Security (CNS), IEEE, 2019. doi:10.1109/cns.2019.8802743

[21] T. Zhyrova, et al., Benchmarking between the DQL index and the web application accessibility index using automatic test tools, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3288, 2022, 110–116.
[22] A. Cheung, A. Solar-Lezama, S. Madden, Optimizing database-backed applications with query synthesis, in: 34th ACM SIGPLAN conference, ACM Press, 2013. doi:10.1145/2491956.2462180
[23] L. Kupershtein, H. Lutsyshyn, M. Krentsin, Information technology of software data security monitoring, Cybersecur. 3(23) (2024) 71–84. doi:10.28925/2663-4023.2024.23.7184
[24] V. Balatska, V. Poberezhnyk, I. Opirskyy, Utilizing blockchain technologies for ensuring the confidentiality and security of personal data in compliance with GDPR, in: Cyber Security and Data Protection, vol. 3800, 2024, 70–80.
[25] Y. Dreis, et al., Model to formation data base of internal parameters for assessing the status of the state secret protection, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, CPITS, vol. 3654 (2024) 277–289.
[26] Y. Dreis, et al., Model to formation data base of secondary parameters for assessing status of the state secret protection, in: Cyber Security and Data Protection, vol. 3800 (2024) 1–11.
[27] M. Jegelka, Secure coding—challenges in information security, 2023. URL: https://www.dqsglobal.com/intl/learn/blog/secure-coding-challenge-in-information-security
[28] O. Vakhula, I. Opirskyy, Research on security as code approach for cloud-native applications based on Kubernetes clusters, in: Cyber Security and Data Protection, vol. 3800, 2024, 58–69.
[29] S. Thomas, L. Williams, T. Xie, On automated prepared statement generation to remove SQL injection vulnerabilities, Inf. Softw. Technol. 51(3) (2009) 589–598. doi:10.1016/j.infsof.2008.08.002
[30] Y. Momryk, Numeric key types for designing optimized databases and protecting against SQL attacks, Mod. Inf. Secur. 60(4) (2024). doi:10.31673/2409-7292.2024.040010
[31] R. A. Teimoor, A review of database security concepts, risks, and problems, UHD J. Sci. Technol. 5(2) (2021) 38–46. doi:10.21928/uhdjst.v5n2y2021.pp38-46