A method for implementing security-as-code in relational databases*

Yaroslava Momryk[1,†], Yuriy Yashchuk[2,†], Roman Tuchapskyy[3,†], and Ivan Opirskyy[1,*,†]

[1] Lviv Polytechnic National University, 12 Stepan Bandera str., 79013 Lviv, Ukraine
[2] Ivan Franko National University of Lviv, 1 Universytetska str., 79000 Lviv, Ukraine
[3] Pidstrygach Institute of Applied Problems in Mechanics and Mathematics, NAS of Ukraine, 3b Naukova str., 79060 Lviv, Ukraine

Abstract

The design of relational databases (DBs) and software (SW) used to work with them has been analyzed in terms of components that affect information security, where correctness directly determines the level of data protection. Internal threats arising from imperfect design have been identified. Positive practices are described that enable database design and software development in accordance with secure coding principles. It is substantiated that the design stage of a relational database, when normalization is applied, relationships between tables are formed, and integrity constraints are established, is itself a step toward ensuring data protection, which is often overlooked in database security literature. In particular, it is shown how to create foreign key relationships between tables so that the database management system (DBMS) protects data from integrity violations. Additionally, aspects of software development responsible for secure interaction with the database are highlighted in the context of creating reliable and secure code based on practical programming experience. The "secure code" approach is widely applied in software development and auditing as it prevents internal information security threats, which are the most common cause of data leaks. The requirements for applying this approach are included in updated information security standards and professional developers of databases and software are recommended to take them into account. Elements of a professional approach to database design and software construction that ensure protection against both internal errors and external attacks, including secure execution of database operations are considered. In particular, an approach has been developed for performing especially important transactional operations with databases. It combines transactionality with logging, recording not only the results of operations and errors but also status changes that reflect the actions of an authorized user within a session. This approach enables blocking of unacceptable status transitions, monitoring their sequence, and detecting critically dangerous combinations that may indicate a hacker takeover. It ensures timely prevention of internal errors that compromise integrity and malicious actions, as well as restoration of the full picture of events in case of security incidents.

Keywords

Security as Code; data security; database design; software construction; security threats; data integrity; database normalization; database operations resilience.

1. Introduction

In recent years, the number of combined multi step attacks on databases through web applications has increased, combining SQL injection (Structured Query Language injection), XSS (Cross-Site Scripting), and CSRF (Cross-Site Request Forgery), thereby complicating the protection of corporate and fintech environments. At the same time, errors in the design of databases and the software used to work with them create loopholes both for the implementation of such attacks and for violations of data integrity due to internal mistakes. In this regard, it is relevant to present an updated version of the work [1], which provides an extended analysis and adds a new section on secure software methods with the participation of a new co author.

Problem Statement. When the term "information protection" is applied to databases, the first associations are usually with data encryption, query protocols and backups, user passwords and

* CPITS 2026: Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2026, Kyiv, Ukraine
* Corresponding author.
† These authors contributed equally.
sm.slyala@gmail.com (Y. Momryk); yuriy.yashchuk@lnu.edu.ua (Y. Yashchuk); roman.tuch@gmail.com (R. Tuchapskyy); ivan.r.opirskyi@lpnu.ua (I. Opirskyy)
0009-0001-4114-0347 (Y. Momryk); 0000-0003-4935-497X (Y. Yashchuk); 0000-0002-2556-1088 (R. Tuchapskyy); 0000-0002-8461-8996 (I. Opirskyy)

logins, roles and privileges, server protection, and defense against SQL injections. However, data security depends on professionalism at every step: from database design to the development and deployment of software for working with it — and only with effective and high quality execution of these steps the aforementioned protective measures will be truly effective.

This paper substantiates that data protection begins already at the design stage, and therefore the concept of "secure code" should be extended to the design and development of relational databases.

Such an approach ensures not only the reliability of software but also the integrity and security of data at all subsequent stages of processing and storage. The paper argues that data protection should start at the design stage, and therefore the concept of "secure code" can be extended to the design and development of relational databases. This approach ensures not only the reliability of the software but also the integrity and security of data throughout all subsequent stages of processing and storage.

Research Aim and Objectives The purpose of this paper is to analyze the steps of database design, highlighting those aspects that are directly related to or influence data protection, and to collect best practices that enable the use of mechanisms for correct information storage in relational databases and secure subsequent data handling.

To achieve this aim, the study addresses the following key objectives:

- Analyze the steps of relational database and software design and identify those that affect data security.
- Formulate risks that can be avoided or reliability criteria that can be achieved through analysis in terms of secure code.
- Propose design and development methods that are most appropriate to apply based on practical development experience and recommendations of professional developers and researchers.
- In the software development part, describe optimal algorithms of actions to minimize risks concerning various operations and subject areas of design and development.

2. Literature review

Literature encountered in the internet space — as a statistical representation of easily accessible information on this topic, typically analyzes and defines threats to data and provides definitions of database security [2, 3], and also offers recommendations for creating especially protected databases. However, authors do not emphasize that competent execution of ordinary database design steps is already a method of ensuring data protection. A serious threat to data security comes not only from external dangers but also from internal threats caused by incorrect development or use. In this aspect, the following threats arise first and foremost:

1. Integrity threats (destruction and modification of information).
2. Availability threats (blocking and destruction of information).
3. Confidentiality threats (unauthorized access, leakage, and disclosure of information) [4].

To protect a database, it is necessary to protect all components of the complex system, namely:

1. Data in the database.
2. Database management system.
3. All related applications.
4. Physical and/or virtual database server and underlying equipment [5].

Special attention should be paid to the work [6], which shows that Clean Architecture creates prerequisites for secure software development through isolation of business logic and dependency control, although it is not a method of direct data protection.

In [7], a description of the main threats to databases and recommendations for elimination methods is provided. Literature sources such as [8] focus on inventing methods for creating secure databases or propose multi-level access control — authorization and encryption.

In [9], the authors propose the application of the DevOps (Development and Operations) methodology for working with databases in cloud environments and consider this approach as a component of the Security as Code (SaC) concept. As noted by the authors, "this research explores the 'Security as Code' (SaC) approach, which integrates security policies and practices into the development and deployment pipelines of cloud-native applications on Kubernetes clusters".

As a new perspective on the application of the Security as Code approach, time-based testing of changes in CI/CD (Continuous Integration/Continuous Deployment) pipelines can also be considered, where automated vulnerability scanners and regression visualization make it possible to monitor the emergence of new security issues throughout the entire software lifecycle — from development to deployment [10].

There remains a niche examined in [11] — it concerns the application of secure software development methodology, specifically regarding S-SDLC. It is based on checking security requirements at various stages: analysis, design, development, testing, and maintenance. Especially during the first two, since most system vulnerabilities are generated before programming tasks begin.

In the same direction, the authors of [12] worked, conducting research on databases built on the basis of a universal relations model and presenting methods for ensuring integrity based on the Clark-Wilson model.

In the literature, the use of stored procedures in the context of database security is considered one of the key measures for access control and query processing. However, [13] warns that the mere fact of using procedures does not guarantee security — if dynamic SQL expressions are generated inside the procedure, vulnerability may remain. Additionally, [14] includes stored procedures as part of the protection system, along with parameterized queries and data validation. Attention is also paid to the correct choice of database field structure [15], which covers issues of field selection, database structure, data validation methods, and key hiding, which closely corresponds to the context of this work — particularly hiding key fields, data validation, and SQL injection prevention.

The authors of [16] explain the danger of CSRF (Cross-Site Request Forgery) attack vulnerabilities, which are especially dangerous when hackers target an employee with high access privileges.

In recent years, the share of incidents related to CSRF has significantly increased, confirming the need to improve request control mechanisms and multi-factor verification. The growth in the number of web attacks, particularly CSRF, session hijacking [17], and cookie theft, indicates systematic problems in session and token management in web applications. In parallel, combined multi-step attacks are developing that combine SQL injections, XSS, and CSRF.

Among attack vectors, SQL injections pose a particular danger to information systems, allowing attackers to leak internal identifiers and other sensitive data from databases, and subsequently use the obtained information to prepare and implement further attacks [18]. Research confirms that IDOR (Insecure Direct Object Reference) vulnerabilities remain one of the most common security problems in modern web systems [19]. In the context of countering these threats, various approaches have been proposed: authorization and access control mechanisms [20], as well as database integrity verification schemes based on smart contracts [21]. In [22], Opirsky proposed a method for optimizing information security system behavior under external and internal influences.

Reliable design of database structure and software is the first need, and its absence is the first threat. Research shows that design flaws constitute a significant security problem. Specifically, [23] notes that design flaws account for 50% of software security problems, which is also confirmed by McGraw's research [24]. Therefore, our material consistently covers all steps of database development and software for them, highlighting those threat causes that are often embedded at the database and application design stage — "internal threats" that also complicate the creation of protection against external threats.

Markus Yetelka [25] writes: “The high importance of secure coding practices is highlighted in the updated information security standards ISO/IEC 27002 and ISO/IEC 27001:2022. As a new information security measure (control), the principles for ‘Secure Coding’ in software development are entering the catalog of measures in Annex A of ISO/IEC 27001.” This new control emphasizes secure coding practices to prevent vulnerabilities and ensure software security through proper development, implementation, and review processes.

Therefore, this article is intended to be a timely response to the updating of standards and the strengthening of external attack vectors, to find and track all links responsible for data security and not only those directly associated in the literature with security measures.

3. Analyze stages of database development and issue results

3.1. Database design

The concept of protecting data stored in a database is closely related to the term “data protection.” According to Wikipedia, data protection refers to a set of methods and tools that ensure the integrity, confidentiality, and availability of information under the influence of natural or artificial threats, the implementation of which may cause harm to the owners and users of the information.

Therefore, the first stage of data protection during development is proper database design, which ensures data integrity and protection against loss of relevance when working with the database. The main aspects here include normalization and integrity rules embedded during the design of the structure, particularly the types of fields that serve as keys and fields used for external relationships between tables (protection against deletion, insertion, and update anomalies). This includes schemas and relationships between tables in the case of a relational model, as illustrated in Figure 1.

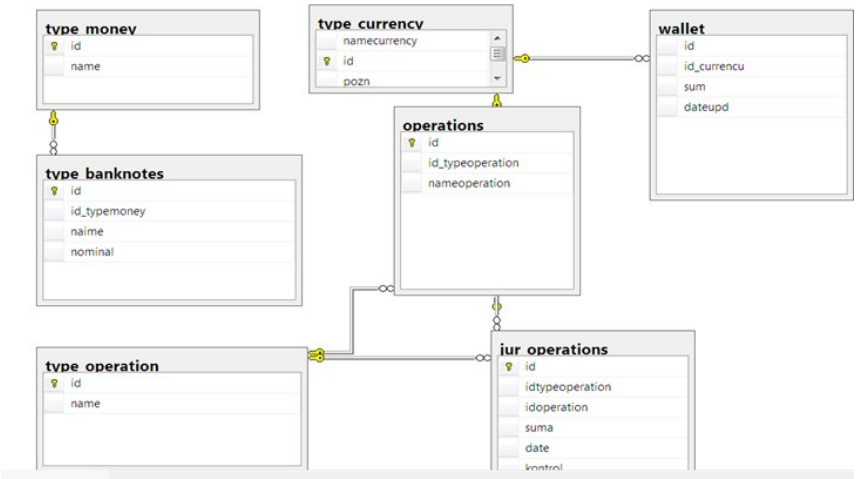

Figure 1: Relational database schema

3.2. Foundations of database protection at the DBMS Level

At the database design stage, the following must be considered: relation integrity, attribute integrity, and referential integrity between relations, as reflected in the database schema. This means that during design (if this is not done correctly, entity integrity will be compromised), it is necessary to select the correct type of key field, ensuring that it is non-null and unique (otherwise the key cannot be created) and has a type suitable for establishing foreign key relationships with child tables.

Moreover, it must be understood that some integrity constraints are established by the developer, and responsibility for data protection begins at this stage. Incorrect constraints that do not match the real requirements of the subject domain will violate integrity, even if enforced. A simple example: if a field requires a length of 300 characters, but only 250 are allowed, some data will be lost.

According to [5], the integrity of relational data is ensured by foreign keys in child relations. A child relation is one that contains foreign keys whose values must refer to the potential key values in the parent relation. Referential integrity is maintained if the value of a foreign key in the child relation refers to an existing value of a potential key in the parent relation or is set to NULL.

The permissibility of using NULL among the attributes serving as foreign keys is determined by user requirements [5]. The reliability of subsequent database operations depends on the correctness of this decision. At this stage, it is necessary not only to choose the correct field type so that the foreign key is created properly but also to consider the possibility of establishing a relationship between tables in a way that ensures the DBMS protects against incorrect operations and data integrity violations. For example, if cascade deletion of related records is enabled when building foreign key relationships, a potential vulnerability in data protection at the DBMS level may be created.

Figure 2 demonstrates how attribute integrity constraints are applied when designing foreign key relationships between tables. The parent field must be either a PRIMARY KEY or UNIQUE, and the parent and child fields must match in type and size. For example, in the tables type_operation and operations, a foreign key relationship is established between id (parent table type_operation) and id_typeoperation (child table operations).

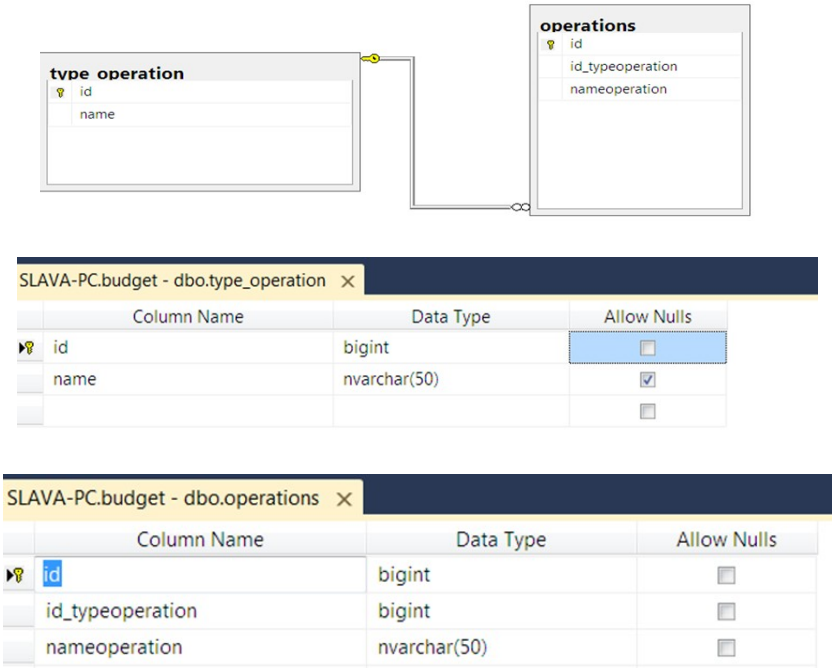

Figure 2: Attribute integrity constraints in the design of foreign key relationships between tables

By defining relationships through foreign keys, the DBMS can be leveraged to prevent incorrect deletion or update of records due to oversight or malicious intent. By default, the DBMS prohibits such actions if they involve a parent record referenced by tuples in child tables. However, the foreign key definition allows specifying alternative behaviors for child table records when a parent record is deleted and/or updated.

This can be illustrated using an example with MS SQL Server tables. A foreign key relationship is established between the tables (Figure 2), and the child table contains data.

Under such constraints, the DBMS will not allow deletion of the parent record. The DBMS generates an error message when an attempt is made to incorrectly delete a parent record that has corresponding child table data.

During the implementation of relationships, it is also possible to prevent incorrect data entry in the child table by making the reference field to the parent table mandatory.

For example, in the operations table, the id_typeoperation field is defined as NOT NULL (see Figure 2). Therefore, if an attempt is made to add a record without a corresponding parent, which

would violate data integrity, the DBMS will issue a warning and prevent the action, as illustrated in Figure 3.

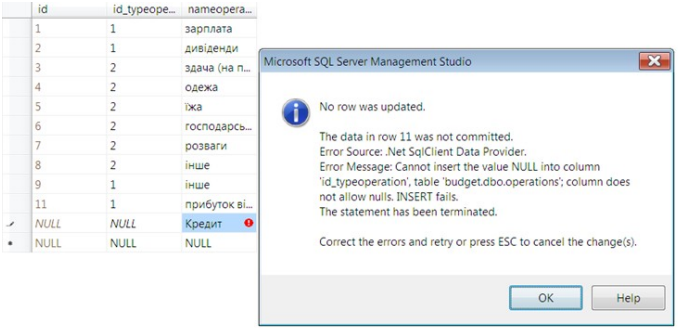

Figure 3: DBMS message when attempting an incorrect insertion of a child record that violates the NOT NULL constraint for a field used as a foreign key to the parent table

3.3. Normalization and relational links as a data protection method

By normalizing the database (to the 2nd and 3rd normal forms) and establishing relationships between tables in a relational database, fields referencing the keys of parent tables are introduced instead of storing the actual data. This approach makes it more difficult to steal information from the database, as the tables contain field codes while the content resides in the corresponding reference tables.

Furthermore, if necessary, when creating tables and fields in the database, their names can be encoded to anonymize them, making it difficult to infer the contents of a table or field. Only the software that interacts with the database presents the data to the user and displays user-friendly field names in the interface.

Figure 4 illustrates an example: a relational database table with encrypted field names containing references to other reference tables. Clearly, the use of relationships between multiple tables helps protect the data from unauthorized viewing and theft.

	REC	F1_	F1_MI	F9_M	F9_MI
1	1802DBF3EA9BB	474CD925B8CE	F8317C73F79B	000000000000	000000000000
2	38925F32099EE	6DCFAB3240D7	C8D749208C9B	50B77DBD5FD	807A15B25052
3	D8AB6A1BA89BE	474CD925B8CE	184DE42BCA9B	50B77DBD5FD	807A15B25052
4	F0730B04A99FB	474CD925B8CE	086E0A6F629C	000000000000	000000000000

Figure 4: Relational database table with encrypted field names

3.4. Internal threats in data retrieval and elimination methods

It is advisable to create views and allow client applications to work with them rather than directly accessing the tables. This provides a dual layer of protection:

- At the access permission stage, the client cannot access the table directly, but only its view.
- Certain fields, such as key fields or those used for data protection, can be hidden from the client to prevent unauthorized viewing.

When retrieving data, very complex queries and views should be avoided, as they may cause timeouts or failures due to prolonged processing. Such retrieval issues create vulnerabilities similar to those described in [4], analogous to DDoS attacks, where a cybercriminal overloads a target server, making it unable to process legitimate requests from real users. In this case, the server becomes unavailable or fails; for problematic data retrievals, this occurs due to internal causes. Therefore, queries should be optimized, and temporary tables or server-side procedures should be used.

When accessing queries from program code, it is also possible to programmatically increase the timeout to prevent abrupt termination of requests, although ideally, queries should not require this. When creating indexes that speed up searches and query execution, it is important to maintain balance: excessive indexing can reduce the speed of data modification operations

(insert/update/delete). Therefore, indexes should be created judiciously, ensuring their maintenance cost does not exceed their performance benefit.

Among best development practices for SQL query optimization is the use of joining selections (subselections) (SQL language operation — JOIN) instead of subqueries. For example, even selection criteria or competency matrices used by software development companies for evaluating candidates typically require knowledge and skills in efficient query formulation.

Consider the following examples:

1. It is necessary to determine the number, manufacturer, and price of the most expensive item; in other words, to find the row containing the maximum value of a certain column. This should not be done using a nested query:

```
SELECT kod_article, produser, price
FROM order
WHERE price=(SELECT MAX(price) FROM order)
```

The same result can be obtained with a better execution plan by using a JOIN:

```
SELECT Max_art.kod_article, dbo.shop. produser, dbo.shop.price
FROM (SELECT MAX(kod_article) AS article
FROM dbo.order) Max_art INNER JOIN
dbo.order ON Max_art.kod_article = dbo.order.kod_article
```

2. How to optimize the count of students for each admission order in the list of orders?

```
SELECT *, (select count(order_id) from [stest].[dbo].[ordernewstudentitem] as item where dbo.
[order].id=item.order_id ) as countitem
FROM dbo.[order]
```

The query with a better execution plan, providing a time gain on a large volume of data:

```
SELECT dbo.[order].*, item.countitem
FROM dbo.[order] INNER JOIN
(SELECT order_id AS id, COUNT(id) AS countitem
FROM dbo.ordernewstudentitem
GROUP BY order_id) AS item ON dbo.[order].id = item.id
```

3.5. Access rights specification and integrity enforcement. Server-side programming

When programming database management and performing any operations on databases through an application, in addition to establishing a secure connection to the database (which is not considered at this stage), it is necessary to specify access rights. At the DBMS level, protection should be configured via user permissions and/or user roles for [20]:

- database creation;
- definition, redefinition, or deletion of tables or views;
- selection, insertion, deletion, or updating of rows in tables or views;
- execution of stored procedures.

The correctness of the modeled application domain and the enforcement of integrity rules determine the proper functioning of data operations and the preservation of data accuracy. An effective tool for controlling the correctness of changes and ensuring integrity — including semantic integrity — at the DBMS level is triggers. They respond to operations and prevent incorrect modifications or perform changes according to integrity rules defined for a specific database, and they also help maintain additional change logs for database administration.

Thus, server-side programming is a critical element of ensuring data security at the DBMS level, encompassing not only triggers but also stored procedures, functions, and sequences. These mechanisms enable centralized implementation of business logic, access control, data integrity enforcement, and uniqueness of identifiers. This approach reduces the risk of malformed queries, enhances system resilience to attacks, and optimizes the handling of critical operations.

Server-side functions and procedures can enforce the logical sequence of changes in the database. Unlike triggers, they are not attached to a table or view but exist as independent objects invoked by an application or another server-side procedure. Stored procedures and functions serve as important mechanisms for access control, centralization of business logic, and reduction of the attack surface — provided they are implemented securely. Access to data via stored procedures and functions increases security, as users can be granted permission to execute a procedure without direct access to tables.

Moreover, stored procedures execute faster because the server compiles them in advance. This reduces the likelihood of errors since the logic is centralized and not duplicated across multiple applications. Using server-side triggers, procedures, and functions, it is important to implement logging of errors and critical events, which enables error correction, auditing, and tracking of potential attack vectors.

Sequences, which are also part of server-side programming in a DBMS, are programmed and used for for the following steps that ensure information protection :

2. Generating unique keys — automatic creation of identifiers without duplication risk;
3. Data integrity control — preventing conflicts when inserting new records;
4. Performance optimization — sequences can be faster than auto-increment mechanisms in some DBMSs.

Proper use of sequences in combination with procedures and triggers helps prevent identifier leakage or forgery.

4. Design and development of database software and secure coding practices for data operations

4.1. Approaches to software construction

After forming the database structure, the next step is developing software that interacts with the database, taking into account security and operation execution control. Application software must complement the security mechanisms implemented at the DBMS level and stored procedures, ensuring reliable execution of data entry, update, and deletion operations.

In particular, application software should also be developed with consideration for operation permissions, for which access rights are specified at the DBMS level. On the software side, integrity constraint compliance for attributes is also monitored by validators — these can be procedures that work both on the client side and server side, as well as ready-made controls (for example, in Bootstrap). Validators, especially server-side ones, not only check data entered into the database for compliance with integrity constraints, but can also check for malicious commands in query parameters sent by the client program. The advisability of parameterized queries and validation to prevent so-called SQL injections is also discussed in [7]. It should be noted that in this sense, it is effective to use prepared templates — placeholders for SQL queries. They can be understood as predefined templates that will be replaced with actual values during execution. The server first processes the SQL expression template, then receives the actual values, validates them, and uses them directly at the moment of query execution (after the query itself has been formed). Thus, the threat of the server executing malicious injections is eliminated.

Another fundamental mechanism for ensuring data integrity is the use of transactions. Transactions allow correct handling of situations where one logical operation consists of several interrelated database queries. If at least one of these queries does not execute correctly, all previous changes must be rolled back. A classic example is order processing and goods shipment operations:

data in the orders, goods movement, and warehouse inventory tables must be updated simultaneously. A consistency violation in at least one of these operations leads to data anomalies.

Transaction requirements, known as ACID principles (atomicity, consistency, isolation, durability), are described in detail in professional literature, so they are not discussed separately within this article. However, it should be emphasized that compliance with ACID principles is a mandatory condition for ensuring data protection and integrity when working with databases.

Software reliability also lies in the application level's ability to anticipate typical errors that the DBMS could detect as messages about integrity constraint violations or lack of access rights. Such situations should preferably be handled at the user interface level, without sending obviously incorrect queries to the DBMS.

The simplest example of such verification is restricting access to functionality for unauthorized users. Users of public internet resources are directed to a login page before performing data operations (for example, placing an order) or offered to complete a registration procedure. Only after authorization does the application program form and send a request to the server to add records to the orders table.

Similarly, if a manager enters product data, the software must not allow creation of a product record without specifying the group to which it belongs. In the absence of the corresponding group, the user is prompted to create it. Thus, at the application software level, foreign key integrity control is implemented between the parent product groups table and the child products table. Formally, this corresponds to a NOT NULL constraint on the foreign key field, but the incorrect query is not sent to the DBMS at all.

A similar approach is applied to deletion operations. If a user attempts to delete a product group for which records already exist in the products or orders tables, the application program must warn about the impossibility of such an action. This is an implementation of foreign key integrity protection at the software level. Otherwise, similar protection would be provided by the DBMS through constraints like ON DELETE RESTRICT (or ON DELETE NO ACTION), as shown in Figure 3, but intercepting the situation at the software level allows providing the user with a clear explanation.

When developing administrative software, for security reasons it is advisable to hide the access path to the administrative part of the application from regular users, even if they have credentials to access the database. It is recommended to additionally protect the administration program itself with a password and, if possible, not display its exact address or name in the browser. Although this article does not aim to provide a detailed examination of web application threats, it should be noted that both client and administrative software require correct routing and protection against unauthorized redirects.

Depending on the database's purpose, a decision is made regarding unauthorized user access to view information. For online stores, such access is usually justified, whereas for internal enterprise databases, even viewing information should be restricted to authorized users only, or access to the corresponding software should be protected. The authors of [22] explain the danger of CSRF (Cross-Site Request Forgery) attack vulnerabilities, which are especially dangerous when hackers target an employee with high access privileges.

Countermeasures against such attacks must be implemented both at the software level and at the execution environment configuration level — data access rights and operations with it should be granted with consideration for the minimal set within the functions that the user performs with the database.

For employees who have access to perform operations on individual database tables, it is advisable to use a role-based access model. For example, roles such as "manager," "accountant," "director" can be defined, for which a list of available tables and permitted operations is specified. Users are assigned to specific roles, which avoids individual access rights configuration for each account.

For authorized users, the application program, according to the chosen architecture, can implement various access control strategies: check the user's role before performing an action,

create different interfaces or menus for different roles, hide buttons or menu items corresponding to operations for which the user does not have permissions.

Important aspects that meet the goal of creating secure code when software interacts with databases are encryption and SQL injection prevention. We will not dwell on this topic in detail, only highlighting some aspects that should be noted.

Regarding encryption, it should be noted that both data encryption itself should be applied (here TDE certificates should be separately noted, which provide protection against database copying), and data traffic encryption — the use of TLS/SSL certificates for data transmission. SQL injection interception is important both for preventing open attacks and for combating "silent" data theft.

Some new technologies and development platforms have certain built-in elements of such protection (for example, LINQ to SQL, HTTPS).

In the sections of the article devoted to software development, no code examples are provided, as the described approaches apply to all software applications for working with databases, regardless of development language and implementation environment. But it should be emphasized: if using easily vulnerable web technology like PHP+MySQL, special attention should be paid to security aspects, particularly protection against SQL injection attacks.

Similarly, modern frameworks incorporate built-in security mechanisms that further reduce vulnerability risks. For example, ASP.NET Razor implements two key approaches:

1. Automatic HTML encoding — by default, converts all data rendered in views into a safe format. This prevents malicious script execution and protects against XSS attacks.
2. Anti-CSRF tokens — a special token is automatically added to forms and verified by the server upon submission. This prevents forged requests and protects against CSRF attacks.

Thus, proper technology selection and its secure implementation are critical factors in ensuring data protection and information system resilience.

Similarly, modern frameworks incorporate built-in security mechanisms that further reduce vulnerability risks. For example, ASP.NET Razor implements two key approaches:

1. Automatic HTML encoding — by default, converts all data rendered in views into a safe format. This prevents malicious script execution and protects against XSS attacks.
2. Anti-CSRF tokens — a special token is automatically added to forms and verified by the server upon submission. This prevents forged requests and protects against CSRF attacks.

Thus, proper technology selection and its secure implementation are critical factors in ensuring data protection and information system resilience.

4.2. Secure algorithms for database operations implementation

Below, we systematize groups of DML (Data Manipulation Language) database operations and the necessary approaches for their secure execution to ensure data protection and integrity.

4.2.1. Connection and data exchange operations between client and server

Database connections and data exchange between client and server application components must be implemented in accordance with the principle of least privilege. Connection parameters should be stored in a separate configuration file.

Connections must be established on behalf of a pre-created DBMS user with a limited set of permissions sufficient only for performing necessary operations. This is a fundamental approach from the perspective of professional software development; detailed authorization and authentication methods are not discussed in this work. Such an approach reduces the consequences of potential attacks and is one of the basic countermeasures against CSRF attacks — a request forgery method where a user unknowingly performs unwanted actions on their behalf. Anti-CSRF tokens, which are verified by the server with each request, are also used to counter such threats.

It is also important to pay attention to cookie protection. To minimize XSS attack risks, the following cookie attribute configuration algorithm is recommended:

1. Set the HttpOnly attribute to block cookie access via client-side scripts (JavaScript). This prevents malicious code injected through XSS vulnerabilities from reading cookies. The exception is cases where the server-side is implemented in JavaScript (for example, Node.js), where this attribute may conflict with legitimate server operations.
2. Enable the Secure attribute to ensure cookie transmission exclusively over secure HTTPS connections, preventing data interception in unencrypted form through unsecured HTTP protocol.
3. Apply the SameSite attribute with Strict value to prevent cookie transmission in cross-site requests (CSRF attacks), restricting their use only within the same domain.

Data transmission between client and server must be carried out using secure communication channels (TLS/SSL).

4.2.2. Data viewing, entry, and update operations

The use of numeric keys combined with mechanisms for hiding actual identifiers (for all types of key fields) and secure data transmission during client-server interaction is justified and corresponds to modern information system design recommendations. Such an approach allows not only optimal use of DBMS resources but also reduces the risk of successful attacks through the application of basic yet effective validation mechanisms. The principles of building such solutions and their effectiveness are substantiated in relevant scientific research, particularly in [14].

Although numeric identifiers are the optimal approach both in terms of resource costs and data protection implementation, there are a number of tasks where the use of UUID-like identifiers is necessary — particularly for distributed system integration or in cases where the number of records exceeds the capabilities of a standard numeric range. The UUID type is used in the standard mechanism for unique object identification in web applications. Its widespread use is due to the possibility of independent generation without collision risk.

When implementing database operations, it should be considered that identifiers often become targets of IDOR-type attacks, where an attacker substitutes the identifier in a request to gain unauthorized access to data. Particularly dangerous are situations where identifiers are transmitted in URLs, query parameters or forms, or used as session identifiers stored in cookies. Identifier disclosure creates potential vulnerabilities for SQL, XSS, CSRF, and IDOR attacks.

As noted in [22], some developers believe that UUIDs can be openly displayed in URL strings. Such an approach can be accepted for tasks like hiding contact data from advertisers, when an identifier is created and transmitted to avoid revealing, for example, an email address. However, for tasks involving authorization, issuing election ballots, or banking transactions, such key data requires primary protection from disclosure. Typically, JSON format, serialization, or graphical QR codes are used to protect UUIDs. In [26], an alternative graphical conversion method is proposed. However, identifier hiding remains the basic method.

Therefore, when implementing data access and viewing, it is especially important to implement the software application in such a way that users can see only the data they are authorized to access, while key fields remain accessible exclusively to administrators. Implementation of such approaches is advisable through server-side programming elements — views and stored procedures.

Data entry and update operations are most vulnerable to SQL injection and IDOR type attacks. Therefore, all data received from users must be validated both on the client side and on the server side. No parameters should be used by the server without prior validation.

Key identifiers necessary for performing update operations should not be displayed openly. For this purpose, hidden form fields or server-side mechanisms for storing identifiers in sessions are used. Before executing an update operation, it is advisable to verify that required fields have not been accidentally cleared or changed to incorrect values [27–29].

As noted in the previous section, parameterized (prepared) queries, in which the SQL expression structure is separated from actual parameter values, are an effective protection mechanism. At the

same time, it is important to consider another aspect: if the input form does not contain correct data or required fields are left empty, the query should not be sent to the server at all, to avoid unnecessary load on the DBMS.

4.2.3. Data deletion operations

Record deletion operations require particularly correct implementation, as user errors or inattention can lead to irreversible data loss. For this reason, it is advisable to apply logical deletion for regular users, whereby a record is transitioned to an inactive state and hidden from standard viewing operations. Physical record removal should be left to administrators.

Before executing a deletion operation, it is advisable to implement protection against accidental invocation of such action — for example, from mistakenly clicking the corresponding button. For this purpose, the application program should request action confirmation and display the data to be deleted. If deletion is to affect only one record, the query should explicitly limit the number of modified rows (for example, using LIMIT 1).

Special attention should be paid to preventing cascading deletion of data needed for history and reports. Before destroying a parent record, the presence of such child data must be verified. For tables where deletion anomalies may occur, cascading deletion through foreign keys should not be used at all. Instead, integrity control and verification of deletion necessity should be implemented at the application software level or through stored procedures.

4.3. Ensuring the resilience of critical operations. Method for controlling status transitions and their combinations

To implement complex database operations consisting of multiple logically connected steps and requiring enhanced protection (to ensure the resilience of critical database operations), it is advisable to use transactions. Combining transactions with server-side procedures ensures atomicity of changes and centralized logging of events, errors, and failures, enabling prompt detection and analysis of system deviations. Other DBMS programming tools are also recommended — for example, triggers can enforce valid state transition sequences and prevent skipping mandatory intermediate states.

Implementing the core for executing such critical operations on the server side using DBMS programming tools, particularly stored procedures with transactions, enable centralized access control, enforcement of business rules, maintenance of data integrity, and modification of business logic without rewriting client application code. This guarantees consistent and secure execution of such operations for all clients. It also reduces the risk of exposing the internal structure of the database and minimizes the attack surface, as the client only provides input parameters without constructing SQL queries independently.

In recent years, attacks targeting user accounts, including session hijacking or impersonation, have posed significant threats. In such cases, an attacker may perform operations on behalf of a legitimate user using credentials that are valid from the DBMS perspective. Traditional access control mechanisms may fail, as the attack does not violate formal authentication procedures [17]. This class of threats requires analyzing not only individual access operations but also the system's overall behavior under destabilizing influences.

To counter such threats, it is advisable to integrate a security algorithm into the described approach that combines transactionality and logging with monitoring of user action sequences. The proposed method not only prevents impermissible status transitions of operations but also records status changes, enabling monitoring of their sequences and blocking unacceptable combinations of transitions.

For each subject area, it is recommended to define a list of permissible transitions and critically impermissible combinations of action statuses for monitoring purposes.

Key aspects of the method:

- Each critical operation sequence is executed within a transaction, and the results, errors, or failures are logged.
- The log records both the previous and the new state of the object.
- A central element of the algorithm is controlling permissible transitions between statuses of objects within the subject domain.
- For each entity, a set of possible statuses and allowed transitions between them is defined.
- Attempts to execute an operation that results in an impermissible transition or an atypical combination of transitions are blocked at the application logic or server procedure level and recorded as suspicious events in the log (with additional reinforced verification of authorization and user identity).

Thus, preventing incorrect status transitions combined with action logging allows the detection of behavioral anomalies and reduces the risk of successful account attacks, even in cases of session hijacking or credential compromise. This approach not only ensures data integrity but also allows reconstruction of a complete event history in case of security incidents and timely prevention of malicious actions.

Consequently, the secure coding approach for ensuring operation resilience is complemented by monitoring user action sequences. The idea of optimizing the behavior of information security systems under external and internal influences [22] aligns with the proposed method of controlling sequences of actions and status transitions of critical operations.

Overall, to illustrate this algorithm within the general approach to secure development, the figure from [15] should be modified to include a server-side programming block, as shown in Figure 5.

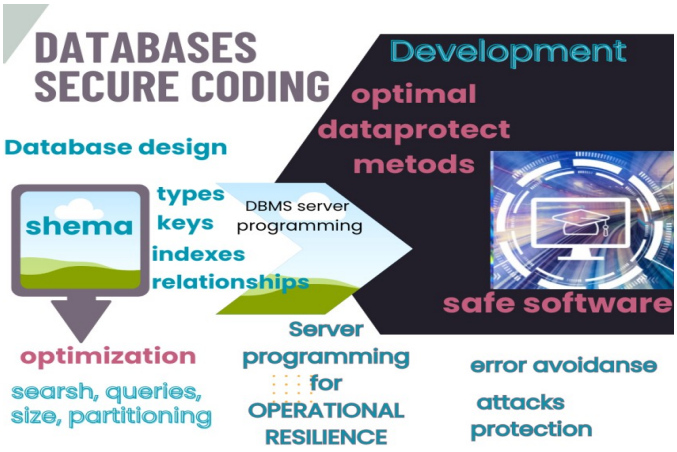

Figure 5: Secure development workflow of databases and software

This reflects the need to integrate competent database design with secure coding practices to ensure data protection.

Conclusions

The paper demonstrates that data protection in relational databases should begin at the design stage. If security foundations are incorporated during the design of the database structure and relationships, then a professional approach at the software design and development stage ensures data integrity, reliability, and protection against invalid database queries at every stage of data processing.

The effectiveness of data protection by the DBMS depends on the professionalism applied in designing relational databases, including normalization, table structures and relationships, field types, integrity constraints embedded in the design, and mechanisms for their enforcement, such as triggers and other server-side programming elements. Only after this can one speak of maintaining integrity and reliability at subsequent stages of software development and operation, including the

development of secure code and the use of specialized data protection mechanisms in the following steps of the software lifecycle.

Therefore, the concept of data protection should be extended to the design of relational databases, and the "secure code" approach should also cover the stages of database design and development. Some developers either fail to recognize this or neglect it, leading to gaps in information security — a manifestation of unprofessional practice. Thus, it is emphasized that a professional approach is necessary to ensure data protection throughout the development of relational databases and the software applications that work with them. The secure code approach, as a component of information security measures for creating certified software products, is recommended not only across all stages of the software development lifecycle but also during the design and development of the databases themselves.

Special attention is given to both internal threats, arising from flaws in database and software development that often create conditions for external attacks, and to modern attack vectors. For critical database operations, a combined approach is proposed that integrates transactional integrity with event logging, not only capturing the results of critical operations and errors but also recording status changes reflecting the actions of an authorized user within a session. It is proposed not only to block unauthorized status transitions, but also to monitor the sequence of status changes and detect critically threatening combinations, which may indicate a hacker takeover. This allows for timely prevention of malicious actions and reconstruction of a complete sequence of events in the case of a security incident.

Thus, the proposed approach extends the traditional understanding of secure code, integrating it into the design and development processes of relational databases and the software that operates with them, and can serve as a practical tool to enhance security for critical database operations requiring special protection.

For future research, it is expected that the material presented will be systematized to formulate criteria to be applied in the design and construction of databases and software, as well as in the assessment of development quality and resilience to security threats.

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] 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, Cybersecur. Educ. Sci. Tech. 3 (23), 2024, p. 42-55, doi:10.28925/2663-4023.2024.23.4255

[2] H. Omotunde, M. Ahmed, A Comprehensive Review of Security Measures in Database Systems: Assessing Authentication, Access Control, and Beyond, Mesopotamian J. Cybersecur., 2023, p. 115-133, doi:10.58496/MJCSC/2023/016

[3] Ş. Mariuţa, Principles of Security and Integrity of Databases, Procedia Econ. Financ., 15, 2014, p. 401-405, doi:10.1016/S2212-5671(14)00465-1

[4] N. Kasianchuk, L. Tkachuk, Information protection in databases, In: Conference Proceedings of "XLVIII Scientific and Technical Conference of Divisions of Vinnytsia National Technical University (2019)".

[5] A. Mousa, M. Karabatak, T. Mustafa, Database Security Threats and Challenges, In: 8th International Symposium on Digital Forensics and Security (ISDFS), IEEE, Jun. 2020, 1-5, doi:10.1109/ISDFS49300.2020.9116436

[6] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Prentice Hall, 2018.

[7] R. Teimoor, A Review of Database Security Concepts, Risks, and Problems, UHD J. Sci. Technol., 5 (2), 2021, p. 38-46, doi:10.21928/uhdjst.v5n2y2021
[8] O. Faragallah, E.-S. El-Rabaie, F. El-Samie, A. Sallam, H. El-Sayed, Multilevel Security for Relational Databases, Auerbach Publications, 2014, doi:10.1201/b17719
[9] O. Vakhula, I. Opirskyy, Research on security as code approach for cloud-native applications based on Kubernetes clusters, Cyber Secur. Data Prot., 2024, pp. 58-69.
[10] I. Zhuravchak, Y. Momryk, O. Vakhula, Time-based testing of changes in CI/CD pipelines for security monitoring, Cybersecur. Educ. Sci. Technol., 4 (24), 2025.
[11] Secure software development methodologies, Between Technology Blog.
[12] V. Yesin, M. Karpinski, M. Yesina, V. Vilihura, K. Warwas, Ensuring Data Integrity in Databases with the Universal Basis of Relations, Appl. Sci., 11 (18), 2021, 8781, doi:10.3390/app11188781
[13] A. Kolhe, P. Adhikari, Injection, Detection, Prevention of SQL Injection Attacks, Int J Comput Appl, 87 (7), 2014), pp. 40-43, doi:10.5120/15224-3739
[14] R. Mahesh, S. Chellathurai, M. Thirunavukkarasu, P. Raman, SQL Injection Attack Detection and Prevention Based on Manipulating the SQL Query Input Attributes, Comput. Sci. Sustain. Technol., Commun. Comput. Inf. Sci., 1973, 2024, pp. 213-221, doi:10.1007/978-3-031-50993-3_17
[15] Y. Momryk, D. Sabodashko, Numeric Fields in Database Development: From Optimal Design to Secure Coding — SQL Attacks Protection Method, Cybersecurity Providing in Information and Telecommunication Systems, 2025, 3991 pp. 84-96.
[16] P. Khurana, P. Bindal, Vulnerabilities and defensive mechanism of CSRF. Int. J. Comput. Trends Technol., 13(1), 171-174, 2014, doi:10.14445/22312803/IJCTT-V13P135
[17] I. Ogundele, A. Akinade, H. Alakiri, A. Aromolaran, B. Uzoma, Detection and Prevention of Session Hijacking in Web Application Management, IJARCCE 9 (7), 2020, pp. 1-10. doi:10.17148/IJARCCE.2020.9601
[18] M. Vaseghi Panah, N. Khatoon Bayat, A. Asami, M. Asadi Shahmirzadi, SQL Injection Attacks: A Systematic Review.
[19] P. Pratama, A. Rhusuli, Penetration testing on web application using insecure direct object references (IDOR) method. In: 2022 International Conference on ICT for Smart Society (ICISS), pp. 1-6. IEEE, Bandung, Indonesia, 2022, doi:10.1109/ICISS55894.2022.9915126
[20] A. Mohamed, D. Auer, D. Hofer, J. Küng, A Systematic Literature Review of Authorization and Access Control Requirements and Current State of the art for Different Database Models, Int. J. Web Inf. Syst., 20 (1), 20241-23, doi:10.1108/IJWIS-04-2023-0072
[21] S. He, X. Xing, G. Wang, Z. Sun, A Data Integrity Verification Scheme for Centralized Database Using Smart Contract and Game Theory, IEEE Access 11, 2023, 59675-59687, doi:10.1109/ACCESS.2023.3284850
[22] Harasymchuk, I. Opirskyy et al., Intelligent Cyber Defence Systems: Detection of Ransomware and Protection of Wireless Networks based on Artificial Intelligence Technologies: Collective Monograph, Kharkiv: Technology Center PC, 2025, 182, doi:10.15587/978-617-8360-22-1
[23] I. Arce et al., Avoiding the Top 10 Software Security Design Flaws.
[24] G. McGraw, Software security : building security in, Addison-Wesley, 2006.
[25] M. Jegelka, Secure Coding, Challenges in Information Security, 2023.
[26] Y. Momryk, Approaches to Secure Data Storage and Processing Based on Uuid Identifiers: Graph. Convers. Method, Cybersecur. Educ. Sci. Tech., 3 (31), 2025, 140-154, doi:10.28925/2663-4023.2025.31.1010
[27] 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.
[28] S. Rzaieva, et al., Methods of modeling database system security, in: Cybersecurity Providing in Information and Telecommunication System, vol. 3654, 2024, 384–390.
[29] S. Zhebka, et al., Method for Adaptive Allocation of Cryptographic Resources in Distributed Databases, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3991 (2025) 620–628.