TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 68, Issue I, March, 2025

A RELATIONAL IMPLEMENTATION WITH WEB INTERFACE FOR STORING ROBOT PROGRAMS

Tiberiu Alexandru ANTAL

Abstract: *The paper is using the XAMPP open-source cross-platform web server solution to develop using the relational database model a free solution to store different categories of robots used in different type of activities together with their associated programs. As XAMPP integrates a web server as well as relational database server the application provides a web page based interface for database interactions.*
Key words: *function, manipulator, parameter, PHP, reuse, routine.*

1. INTRODUCTION

The modern industrial and robotic applications are organized around databases which play an essential role in efficiently managing robot programs. The database provides a centralized location for storing robot programs, ensuring they are easily accessible to authorized users. It allows multiple users or systems to retrieve and update programs simultaneously, enabling collaboration across teams. While from a structural point of view serial robots have well known architectures, parallel robots still support innovation in terms of new designs[1-3], adaptable to specific applications [4-5]. As robotic systems become increasingly complex and interconnected [6] [7], the need for organized, scalable, and secure storage solutions has grown and researches expanded these concepts to database management systems integrated in cloud [8], [9] for system efficient data management and processing in enhancing robotic functionalities and for centralized database to store and process data collected by robots [10], enhancing efficiency and scalability in data center management. While cloud solutions provide robust environments for production, scalability, and collaboration there are also good tools for local development and small-scale testing. Many developers start with XAMPP during development and migrate to cloud solutions for deployment. The following approach is based on **XAMPP** the open-source software package that provides a local web server environment for developing, testing, and running web applications. The name is an acronym that stands for:

- **X**: Cross-platform (works on Windows, macOS, and Linux)
- **A**: Apache (a web server)
- **M**: MySQL (a database system, now often replaced by MariaDB)
- **P**: PHP (a server-side scripting language)
- **P**: Perl (another scripting language)

The MariaDB database system is using the relational data [11] model to manage the data and as well as how the data in organized in the system. In mathematics a data model is defined by the formalism used to describe the data structures and the set of operators used to validate and manipulate the data structures. The relational data model is a strict one which means that all data must be modelled using predefined categories called types. This approach is used in systems that prioritize data integrity, consistency, and strong typing over flexibility. In a relational database management system (RDMS) the structure defines how data is organized (e.g., tables, objects, or graphs). the

Received: 29.01.25; **Similarities**: 05.03.25: **Reviewed**: 20.02./19.02.25: **Accepted**:06.03.25.

relationships specify how different pieces of data are connected and the constraints describe rules that enforce data integrity and validity. In RDMS the data structure used to store the data is the table and the relational algebra operators are used to manipulate the data. A query is a request for information or data from a database. It is a structured way of asking a database to retrieve, insert, update, or delete data based on specific conditions. Queries are typically written in query languages, such as SQL [17], [18] (Structured Query Language). The relational algebra operators can be divided into fundamental and derived categories. These operators manipulate relations (tables) to produce new relations. SQL (Structured Query Language) and Relational Algebra are closely related. SQL is the implementation of the relational model, while relational algebra provides the theoretical foundation. Both use operators to manipulate and query data, though their syntax and focus differ [17], [18]. Table 1 is showing briefly the equivalence between the relational algebra operators and the SQL language.

Table 1

The relational algebra - SQL equivalence.

Operator / Symbol	Meaning	Syntax	SQL Equivalent
Fundamental operators (core operators to build queries)			
Selection / σ	Filters rows based on a condition.	$\sigma_{condition}(R)$	SELECT ... FROM ... WHERE
Projection / π	Selects specific columns.	$\pi_{columns}(R)$	SELECT column_names
Cartesian Product / ×	Combines every row of one relation with every row of another.	R×S	SELECT * FROM Table1, Table2
Union / ∪	Combines tuples from two relations.	R∪S	UNION
Set Difference / −	Finds tuples in one relation but not in another.	R−S	EXCEPT
Rename / ρ	Renames a relation or its attributes.	$\rho_{newName}(R)$	
Derived operators (built using combinations of fundamental operators and are used to simplify queries)			
Intersection / ∩	Finds common tuples between two relations.	R∩S constructed as R−(R−S)	INTERSECT
Natural Join / ⋈	Combines relations based on common attributes.	R⋈S constructed as $\pi_{attributes}(\sigma_{condition}(R\times S))$	NATURAL JOIN
Theta Join / $\bowtie_\theta$	Combines relations based on a condition θ.	$R\bowtie_\theta S$ construted as $\sigma_\theta(R\times S)$	Join with ON or WHERE clause.
Division / ÷	Finds tuples in one relation that match all tuples in another.	R÷S constructed as $\pi_A(R)-\pi_A((\pi_A(R)\times S)-R)$	Simulated with nested queries.

2. CONCEPTUAL MODEL and LOGICAL DATA MODEL

At the conceptual level, the problem to be solved is related to a database used to store robots and robot programs. CRUD [16] (Create, Read, Update, and Delete) must be provided with a proper web interface to facilitate inserting, viewing, updating and changing the information from the database.

The application should be able to store several robots of different types as well as several distinct program associated to each robot depending on the specific tasks the robot should handle at different moments. An Entity-Relationship Diagram (ER Diagram) is a graphical representation of the entities/concepts/tables, relationships/associations, and attributes/ characteristics/columns that define a blueprint for relational database to be designed and are specific to the conceptual design phase when defining and understanding the relationships between data entities.

The ER diagram and database schema are closely related, but they serve distinct purposes in the process of database design and implementation. The database schema is specific to the logical and physical design when the database is created together with the tables, data types, constraints, and indexes. XAMPP integrates a tool called phpMyAdmin which is a web-based tool that simplifies the management of databases, including the design and implementation of a database schema. Using the *Designer* tab from phpMyAdmin in Figure 1 the database schema is given.

The following tables have undergone the normalization [17], [18] process to reduce redundancy and improve data integrity. Normalization involves dividing a database into smaller, related tables and defining relationships between them to ensure that each piece of data is stored only once and at the same time helps eliminate anomalies during data operations such as insertion, update, and deletion.

A Primary Key (PK) is a column or a combination of columns in a table that uniquely identifies each row in that table. A Foreign Key (FK) is a column or a combination of columns in one table that refers to the PK in another table. It establishes a relationship between the two tables.

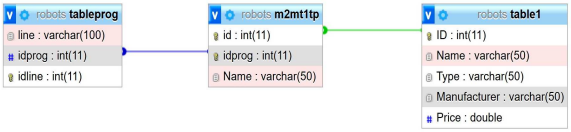

Fig. 1. The database schema.

The schema shows there tables *tableprog*, *m2mt1tp*, *table1* and two relationships. The tables are defined by the attributes/columns as follows (the column data types can be seen in Figure 1. on the right side of the column names like ID: int(11)):

- *tableprog*: line, idprog, idline (PK);
- *m2mt1tp*: id(PK, FK), idprog(PK, FK), Name;
- *table1*: ID (PK), Name, Type, Manufacturer and Price.

The *table1* stores the robots and their type and price with the primary key (PK) **ID**. The table *m2mt1tp* stores the names of the programs associated to a robot, and as a robot can run more programs the relationship between table *table1* and *m2mt1tp* is 1 to many (1:M).
The PK of *m2mt1tp* is composed from **id** and **idprog** at the same time ***id*** is a FK for table1. The *tableprog* table stores program lines from a program having the PK as **idline** and the FK ***idprog***. As a program is made up from more lines a 1 to many relationship (1:M) is formed between the *m2mt1tp* and *tableprog*.

2.1 HTML in database data input and output

Using HTML [14] for data input/output with a MariaDB database typically involves building a web application that connects a user interface (UI) to the database. This allows users to input data via forms and retrieve or display data dynamically in web pages.

The frontend [16] is handling the data input using HTML forms while the output is based on tables. The backend [14], [17], [18] is based on the PHP server-script language [12], [13], [15]. The database (MariaDB) is used for storing, retrieving, and managing the data.

2.2 The ‘Main’ menu of the frontend

The start form based only on HTML (no PHP) is acting as a main menu (see Figure 2), and the corresponding HTML code follows.

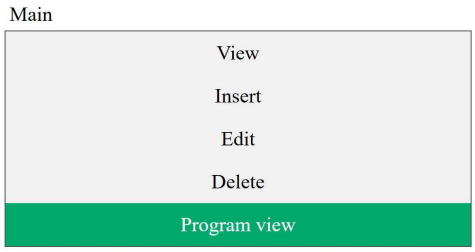

Fig. 2. The main menu of the frontend.

```
<html lang="en">
  <head> <style>
ul { list-style-type: none; margin:
0;padding: 0; width: 25%; background-
color: #f1f1f1; position: fixed;
  overflow: auto; border: 1px solid
#555;}
li { display: inline; text-align:
center; border-bottom: 1px solid #555;
}
li:last-child { border-bottom: none; }
li a {display: block; color: #000;
padding: 8px 16px; text-decoration:
none;}
li a.active { background-color:
#04AA6D; color: white; }
li a:hover:not(.active) { background-
color: #555; color: white;
}
</style> </head> <body>
<table>
<th><td>Main</td></th><tr><td> <ul>
  <li><a href="view.php"> View
</a></li> <li><a href="insert.html">
Insert </a></li> <li><a
href="edit.php"> Edit </a></li>
  <li><a href="delete.php"> Delete
</a></li> <li><a class="active"
href="prog/programsview.php"> Program
view </a></li> </ul> </td></tr>
</table> </body> </html>
```

2.3 The 'View' menu entry

The 'View' menu entry is shown in Figure 3 and is based on the "view.php" code showing the content of *table1*. All the fields/attributes/columns of *table1* are visualized in a HTML table showing the ID (PK), Manufacturer (of the robot), Name, Price and Type. The code is executed on the Apache web server from XAMPP and as it contains the `<?php ... ?>` tag it must have the `php` extension [13], [15] so the sever-side script would be passed to the PHP interpreter to generate the HTML table as the output.

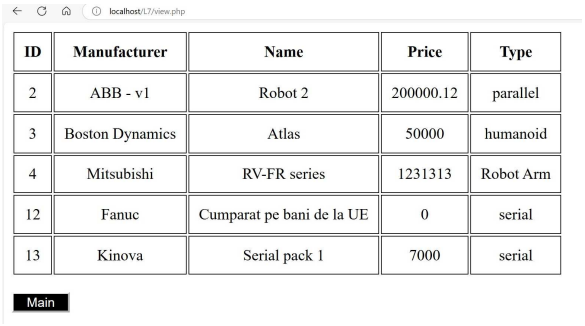

Fig. 3. The 'View' menu entry frontend.

The code is based on a selection using the SQL `"SELECT * FROM Table1"` statement and returns a dynamic output based on the data stored in *table1*.

```
<?php
    include_once 'con_roboti.php';

  $sql = "SELECT * FROM Table1";
  $result = mysqli_query($conn, $sql);

    if (mysqli_num_rows($result) > 0)
    {
      $row=mysqli_fetch_all($result,
MYSQLI_ASSOC);
    }
?>

<!DOCTYPE html>
<html>
<style>
    td,th {
        border: 1px solid black;
        padding: 10px;
        margin: 5px;
        text-align: center;
    }
</style>
<title>Web page to show data form
MariaDB:roboti;Table:Table1 -
view.php</title>

<body>
 <table>
  <thead>
   <tr>
    <th>ID</th> <th>Manufacturer</th>
    <th>Name</th><th>Price</th>
    <th>Type</th> </tr>
  </thead>
 <tbody>
 <?php
  if(!empty($row))
   foreach($row as $rows) { ?>
   <tr><td><?php echo $rows['ID'];
?></td>
```

```
 <td><?php echo $rows['Manufacturer'];
?></td> <td><?php echo $rows['Name'];
?></td> <td><?php echo $rows['Price'];
?></td> <td><?php echo $rows['Type'];
?></td> </tr> <?php } ?>
 </tbody></table>
 <p>
 <form action="main.html">
  <input type="submit" value="  Main  "
style="background-color:black;
color:white; border-color:white"/>
 </form> </body> </html> <?php
    mysqli_close($conn);
?>
```

2.4 The ‘Insert’ and ‘Edit’ menu entries

The ‘Insert’ and ‘Edit’ menu entries are shown in Figure 4 to Figure 6. The frontend of the ‘Insert’ entry is a HTML form that provides the data to the SQL `"INSERT INTO Table1 (Manufacturer,Name,Price,Type) VALUES ('$man','$nam','$pri','$typ')"` statement responsible to insert the data in *table1*.

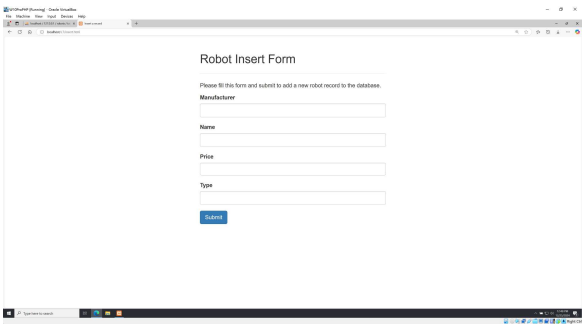

Fig. 4. The ‘Insert’ menu entry frontend.

The ‘Edit’ menu involves a two-step action. The frontend starts in Figure 5 where the robot to be edited must be selected using the left radio button based on the code from the “edit.php” file. The backend is provided as the form from Figure 6, based on the ‘formedit.php’ code, and allows editing and updating the data in *table1*. The connection between the frontend and backend forms is made by the ID (PK) of the row selected in Figure 5 which is transmitted as a parameter to the ‘formedit.php’ code that outputs the record to be edited in Figure 6.

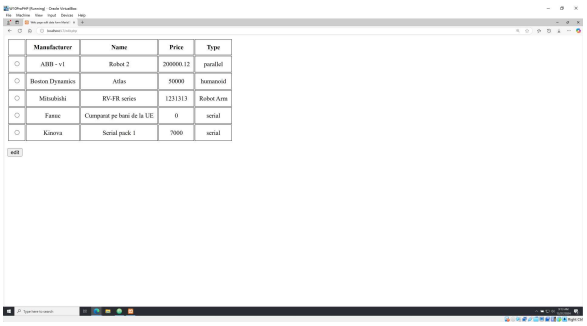

Fig. 5. The ‘Edit’ menu entry frontend.

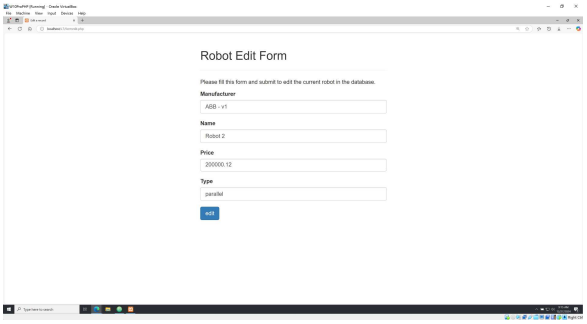

Fig. 6. The ‘Edit’ menu entry backend.

When the ‘edit’ button (see Figure 6) is pressed the code will update the modified data of the row identified by the ID (PK).

2.5 The ‘Delete’ menu entry

The ‘Delete’ menu entry is shown in Figure 7 and is generated by the ‘delete.php’ code. Compared to previous menus, where only one operation was possible with a single row in the table, in this menu it is possible to simultaneously mark several rows (using checkboxes) for deletion and delete them by pressing the ‘Delete’ button. This means that multiple IDs need to be sent from the frontend to the delete code 'del.php'. In the examples so far in the frontend, the interface elements stored a single value in a single name, however in this case several ID values will be stored in the **`'chechbox[]'`** array name (see Figure 7). The **`[]`** notations transforms the scalar variable named **`'chechbox'`** to an array.

The 'del.php' code in the backend is using a for loop with the number of elements extracted in the `$count` variable to delete one by one each of the rows with the IDs stored in the **`'chechbox'`** variable.

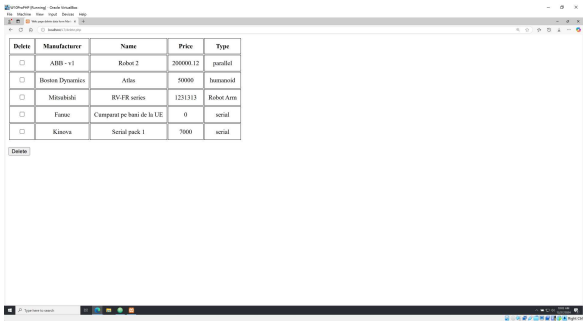

Fig. 7. The 'Edit' menu entry frontend.

```
<?php
    include_once 'con_roboti.php';
    $sql = "SELECT * FROM Table1";
    $result    =    mysqli_query($conn,
$sql);
    if              ($count              =
mysqli_num_rows($result) > 0)
    {
      $row=mysqli_fetch_all($result,
MYSQLI_ASSOC);
    }
?>
<!DOCTYPE html>
<html>
<title>Web   page   delete   data   form
MariaDB:roboti;      Table:Table1      -
delete.php</title>
<body>
<form action='del.php' method='post'>
    <table><thead>                  <tr>
<th>Delete</th> <th>Manufacturer</th>
         <th>Name</th> <th>Price</th>
         <th>Type</th>  </tr>  </thead>
<tbody>
<?php if(!empty($row))
  foreach($row as $rows){?> <tr>
    <td>     <input     type='checkbox'
name='chechbox[]'   value='<?php   echo
$rows["ID"]; ?>'/> </td>
    <td><?php                          echo
$rows['Manufacturer']; ?></td>
    <td><?php    echo    $rows['Name'];
?></td>
    <td><?php    echo    $rows['Price'];
?></td>
    <td><?php    echo    $rows['Type'];
?></td>
     </tr>
      <?php } ?></tbody></table>

<!-- DELETE BUTTON -->
  <p><input type='Submit'  id="Delete"
value='Delete' name='Delete' />
 </form> </body> </html>
<?php  mysqli_close($conn); ?>

<?php
include_once 'con_roboti.php';
echo var_dump($_POST['chechbox']);
if(isset($_POST['Delete']))
{ $count= count($_POST['chechbox']);
$checkbox= $_POST['chechbox'];
for($i=0;$i<$count;$i++)
{ $del_id = $checkbox[$i];
$del   =   "DELETE   FROM   Table1   WHERE
ID='$del_id'";
$result = mysqli_query($conn, $del);
}
if($result)
{  echo  "<meta  http-equiv=\"refresh\"
content=\"0;URL=view.php\">"; }}
mysqli_close($conn);
?>
```

2.5 The 'Program view' menu entry

The 'Program view' entry is based generated by the 'programsview.php' code which is stored in the *prog* subdirectory of the application and integrates all the actions related to the programs associated with a robot using links. In Figure 8 by selecting the proper link we can delete, add or view programs associated to robots.

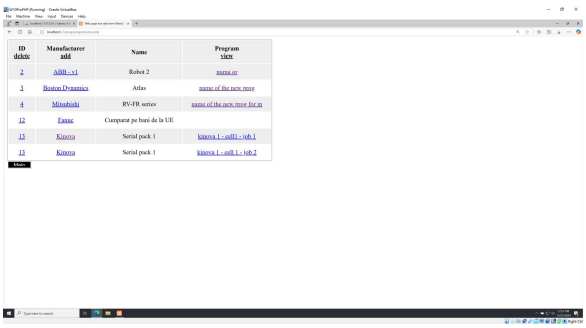

Fig. 8. The 'Program view' menu entry frontend.

Fig. 9. The 'Program view/add' menu entry frontend.

To add a program to a robot the link with the 'Manufacturer' name (the second column from left - see Figure 8) must be selected and the page from Figure 9 is generated. Here, the name of the program must be given and the text file from the local drive must be chosen. To delete a program

associated to a robot the proper link, containing the program name we wish to erase should be selected from the first column in Figure 8
To view the content of a program file the proper link from the last column in Figure 8 must be selected and the content will be generated in a table as shown in Figure 10. The code corresponding to the 'add' link follows and compared to the rest the codes this has two specific characteristics. It need two parameters from the dynamic frontend in Figure 8, the `ID` of the robot from *table1* and the `idprod` of the program which is going to be processed (with add, delete or view) from *m2mt1tp* and It's using and SQL statement which operates on the two mentioned tables or only one table if there are no program associated yet to the robot (see Figure 8 with ID=12).

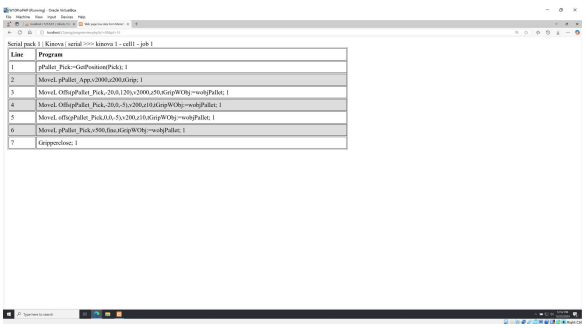

Fig. 10. The 'Program view/add' menu entry backend result.

```
<?php
 include_once 'con_roboti.php';
 $p1 = $_GET['p1']; $p2 = $_GET['p2'];
 $l=1; if ($p1 != null)
  $sqlnmt = "SELECT `table1`.`Name`,
`table1`.`Manufacturer`,
`table1`.`Type`,    `m2mt1tp`.`idprog`,
`m2mt1tp`.`Name`  AS  `program`  FROM
`table1`  LEFT  JOIN  `m2mt1tp`  ON
`m2mt1tp`.`id`  =  `table1`.`ID`  where
`idprog`=".$p1;
  else
   $sqlnmt  =  "SELECT  *  from  table1
WHERE id = ".$p2;
   $result    =    mysqli_query($conn,
$sqlnmt);
   if (mysqli_num_rows($result) > 0) {
    $NMT=mysqli_fetch_all($result,
MYSQLI_ASSOC); }
   echo 'New program will be added to
>>>>    '.$NMT[0]['Name'],'    |    ',
$NMT[0]['Manufacturer'],      '      |
',$NMT[0]['Type'] ;
?>
```

```
   <form    action="programupload.php"
method="post" enctype="multipart/form-
data">       <input       type=hidden
name="idrobot" value=<?php echo $p2;
?>>  <input  type=hidden  name="idp1"
value=<?php echo $p1; ?>>
   <table width="" height="" border="2"
align="left">
  <tr>  <td  width="">Prog  name:</td>
<td><input    type=text    name="pname"
value="name  of  the  new  prog"></td>
/tr><tr> <td width="">Upload File (.txt
only):   </td>   <td   width=""><input
type="file" name="upload"></td></tr>
  <tr><td          colspan="3"><button
type="submit">Upload</button></td></tr
> <tr><td colspan="3">
    <?php    if(!empty($_GET['msg'])){
  echo $_GET['msg'];}?></td></tr>
 </table>
 </form>
```

3. CONCLUSIONS

XAMPP is an excellent package for the development of a robotic related database. It provides tools for the **Frontend** using HTML forms; the **Backend** using the server processes with SQL commands for data interaction and the **Database** to store and retrieve data.

4. REFERENCES

[1] Vaida, C., Pisla, D., Schadlbauer, J., Husty, M., Plitea, N. *Kinematic analysis of an innovative medical parallel robot using study parameters*. In New Trends in Medical and Service Robots. Mechanisms and Machine Science; Wenger, P., Chevallereau, C., Pisla, D., Bleuler, H., Rodić, A., Eds.; Springer: Cham, Switzerland, 2016; Volume 39.

[2] Pisla, D.; Plitea, N.; Videan, A.; Prodan, B.; Gherman, B.; Lese, D. *Kinematics and design of two variants of a reconfigurable parallel robot.* In Proceedings of the ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots, London, UK, 24 July 2009.

[3] Klimchik, A., Chablat, D., Pashkevich, A. Stiffness modeling for perfect and non-perfect parallel manipulators under internal and external loadings, Mech. Mach. Theory, 79 (2014), pp. 1-28, doi: 10.1016/j.mechmachtheory.2014.04.002

[4] Plitea, N.; Hesselbach, J.; Vaida, C.; Raatz, A.; Pisla, D.; Budde, C.; Vlad, L.; Burisch, A.; Senner, R. *Innovative development of surgical*

parallel robots. Acta Electron. Mediamira Sci. Cluj Napoca 2007, 4, 201–206.
[5] Plitea, N.; Hesselbach, J.; Pisla, D.; Raatz, A.; Vaida, C.; Wrege, J.; Burisch, A. *Innovative Development of Parallel Robots and Microrobots*. Acta Teh. Napoc. Ser. Appl. Math. Mec. 2006, 49, 5–26.
[6] Pisla, D.; Plitea, N.; Gherman, B.; Pisla, A.; Vaida, C. *Kinematical analysis and design of a new surgical parallel robot*. In Proceedings of the 5th International Workshop on Computational Kinematics, Duisburg, Germany, 6–8 May 2009; pp. 273–282.
[7] Reiter, A., Muller, A., Gattringer, H. *On Higher Order Inverse Kinematics Methods in Time-Optimal Trajectory Planning for Kinematically Redundant Manipulators*. *IEEE Trans. Ind. Inf.* **2018**, *14*, 1681–1690.
[8] Vijaykumar S, Saravanakumar S G, *Future Robotics Database Management System along with Cloud TPS*, International Journal on Cloud Computing: Services and Architecture(IJCCSA),Vol.1, No.3, November 2011, https://doi.org/10.48550/arXiv.1112.202.
[9] S. Alirezazadeh and L. A. Alexandre, *Optimal Algorithm Allocation for Single Robot Cloud Systems*, in IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 324-335, 1 Jan.-March 2023, doi: 10.1109/TCC.2021.3093489.
[10] Russo, L.O.; Rosa, S.; Maggiora, M.; Bona, B. *A Novel Cloud-Based Service Robotics Application to Data Center Environmental Monitoring*. Sensors 2016, 16, 1255. https://doi.org/10.3390/s16081255.
[11] E. F. Codd, *A relational model of data for large shared data banks*, E. F. Codd. Commun. ACM 13, 6 (June 1970), 377–387. https://doi.org/10.1145/362384.362685.
[12] Rasmus Lerdorf, Kevin Tatroewith, Bob Kaehms and Ric McGredy, *Programming PHP*, O'Reilly, 2002, p. 507, ISBN 1-56592-610-2.
[13] ANTAL, Tiberiu Alexandru. *A review of the PHP server-side scripting language compared to C, C++ and Java for numerical engineering applications*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 66, n. 1, 2023. ISSN 2393–2988.
[14] ANTAL Tiberiu Alexandru, *Proiectarea paginilor Web cu HTML, VBScript şi ASP - ediţia a II-a*, Editura RISOPRINT, 2006, p.264, ISBN 973-751-349-5.
[15] ANTAL, Tiberiu Alexandru. *Generalization by parameterization with associated arrays, in PHP, in a manipulator computation*. ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 67, n. 1, mar. 2024. ISSN 2393–2988.
[16] ANTAL, Tiberiu Alexandru. *A CRUD implementation in JDeveloper and MS Access of a flat database for storing robot programs.* ACTA TECHNICA NAPOCENSIS - Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING, v. 66, n. 1, May. 2023. ISSN 2393–2988.
[17] Thomas Connolly, Database Systems: *A Practical Approach to Design, Implementation, and Management, 6th edition*, PEARSON INDIA, 6th edition, 2019, p. 227, ISBN-13: 978-93534389.
[18] ANTAL Tiberiu Alexandru, *Microsoft Access 97 şi 2000 în 14 cursuri*, Editura Todesco, 2000, p. 299, ISBN 973-99779-6-0.

.

O implementare relațională cu interfață web pentru stocarea programelor robot

Rezumat. Lucrarea folosește soluția de server web multiplatformă open-source, gratuită, XAMPP pentru a dezvolta folosind modelul bazei de date relaționale o soluție gratuită pentru a stoca diferite categorii de roboți utilizați în activități de diferite tipuri împreună cu programele asociate acestora. Deoarece XAMPP integrează un server web, precum și un server de baze de date relaționale, aplicația oferă o interfață bazată pe pagini web pentru interacțiunile cu baza de date.
Cuvinte cheie: *funcție, manipulator, parametru, PHP, reutilizare, rutină*

ANTAL Tiberiu Alexandru, Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@mail.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.