Додаток А
Графічний матеріал атестаційної роботи

ГЮИК. 50 8140.001
(позначення документу)

ЗАТВЕРДЖЕНО
ГЮИК.50 8140.001 – ЛУ

Дослідження процесів міграції баз даних в корпоративних системах

Графічний матеріал

ГЮИК.50 8140.001 – ЛУ

ЛИСТІВ 7

2019 р.

Міністерство освіти і науки України
Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
керівник атестаційної роботи
проф. Гребеннік І.В.

Дослідження процесів міграції баз даних в корпоративних системах

Графічний матеріал

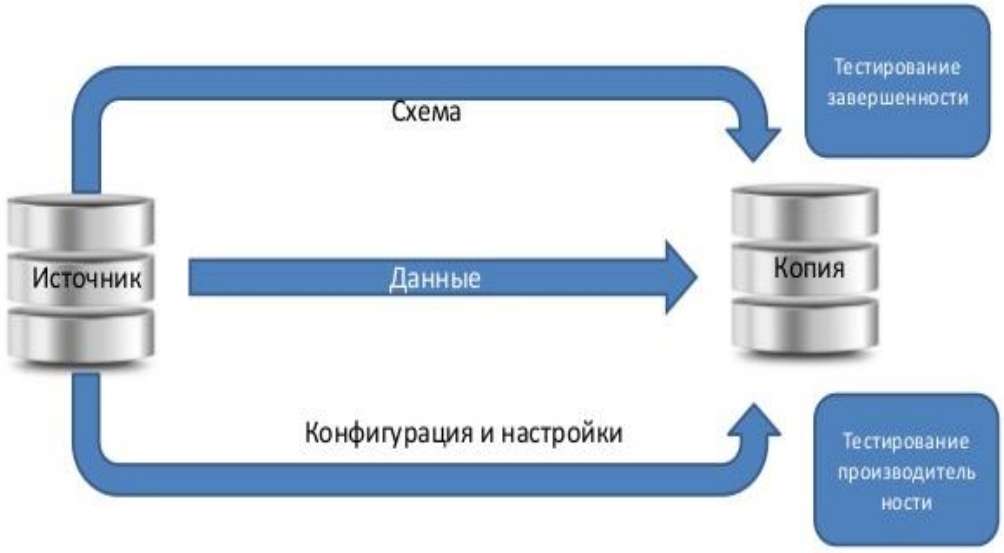
ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК. 50 8140.001 – ЛУ

РОЗРОБИЛА:
ст. гр. ИТПм-18-1
Кошова А.О.

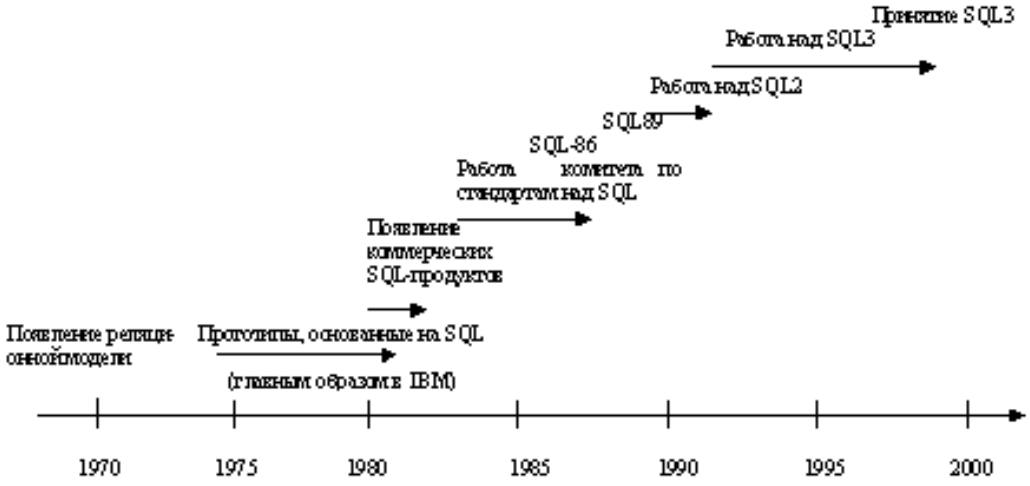
2019 р.

ПРАКТИЧНА МІГРАЦІЯ ДАННИХ



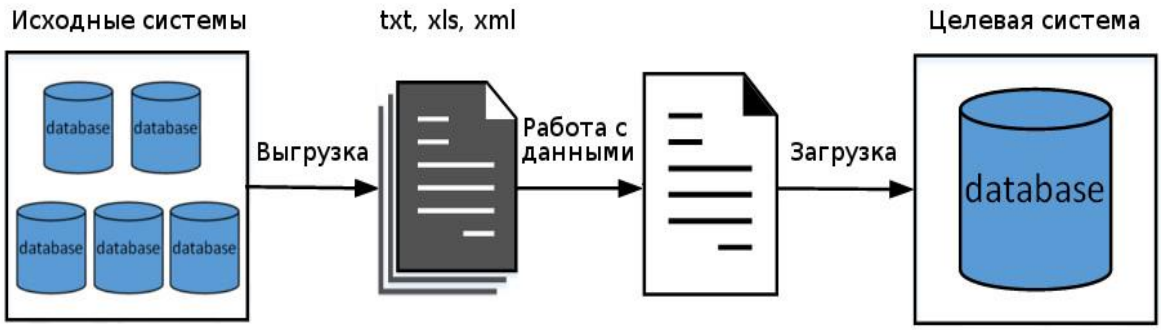
Розроб.	*Кошова А.О.*		18.12	*Практична міграція даних*	
Перевір.	*Гребеннік І.В.*		18.12		
Н. Контр.	*Гребеннік І.В.*		18.12		
Затверд.	*Гребеннік І.В.*		18.12	*СТ*	*Листів 1*

ІСТОРІЯ РОЗВИТКУ МОВИ SQL



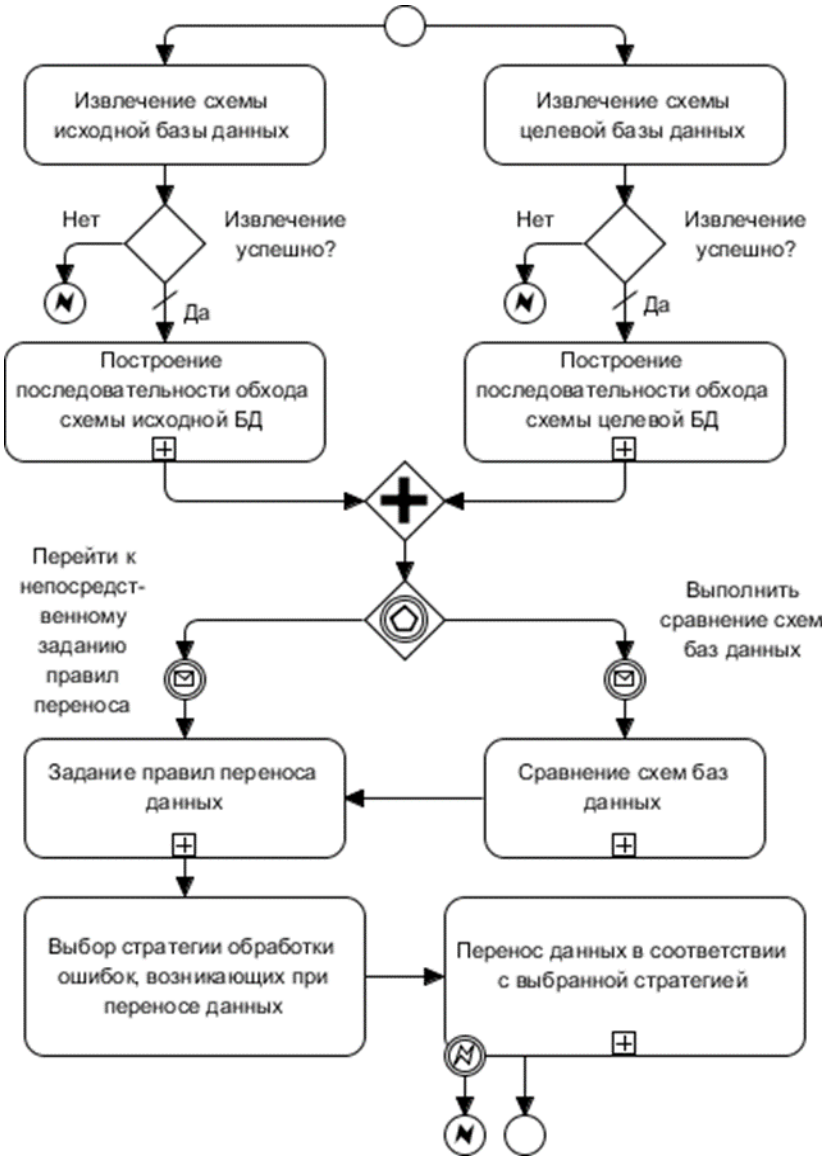
Розроб.	*Кошова А.О.*		18.12	*Історія розвитку мови SQL*	
Перевір.	*Гребеннік І.В.*		18.12		
Н. Контр.	*Гребеннік І.В.*		18.12		
Затверд.	*Гребеннік І.В.*		18.12	*СТ*	*Листів 1*

ПРОЦЕС МІГРАЦІЇ



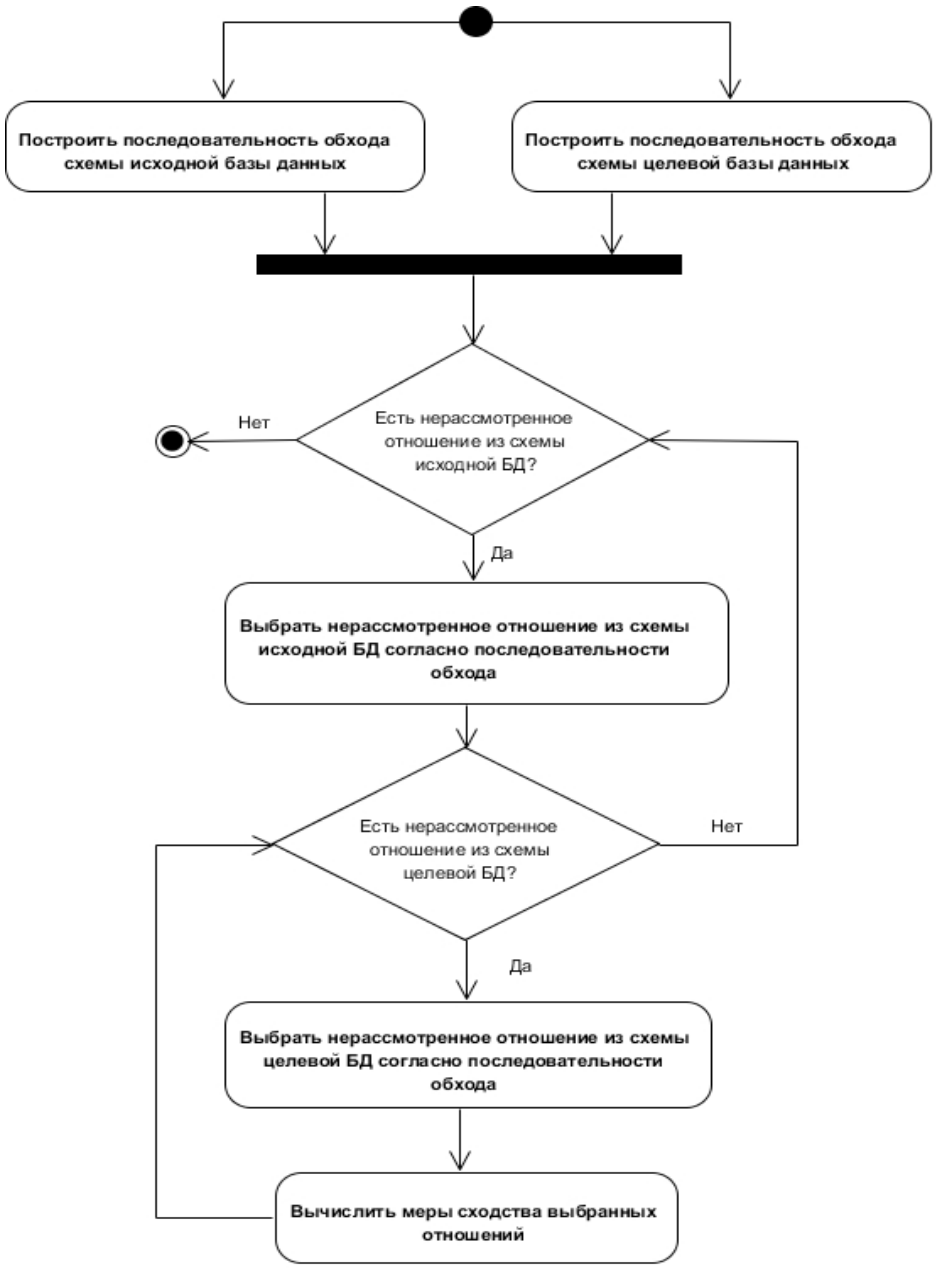
Розроб.	*Кошова А.О.*		18.12	*Процес міграції*	
Перевір.	*Гребеннік І.В.*		18.12		
Н. Контр.	*Гребеннік І.В.*		18.12		
Затверд.	*Гребеннік І.В.*		18.12	*СТ*	*Листів 1*

ДІАГРАМА ПРОЦЕСС МІГРАЦІХ ДАНИХ В НОТАЦІЇ BPMN



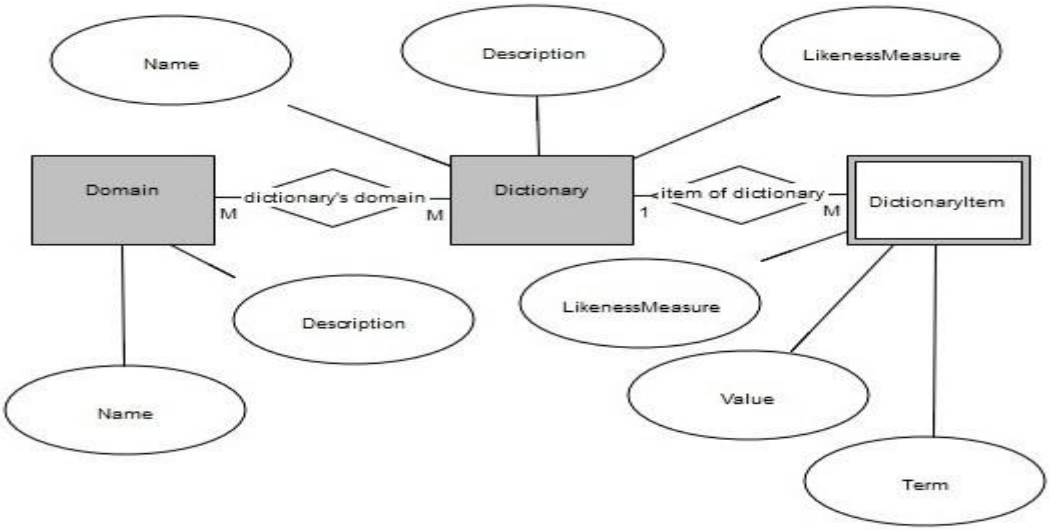
Розроб.	*Кошова А.О.*		18.12	*Діаграма процесс міграції даних в нотації BPMN*	
Перевір.	*Гребеннік І.В.*		18.12		
Н. Контр.	*Гребеннік І.В.*		18.12		
Затверд.	*Гребеннік І.В.*		18.12	*СТ*	*Листів 1*

вДІАГРАМА ПОРІВНЯННЯ СХЕМ БАЗ ДАНИХ



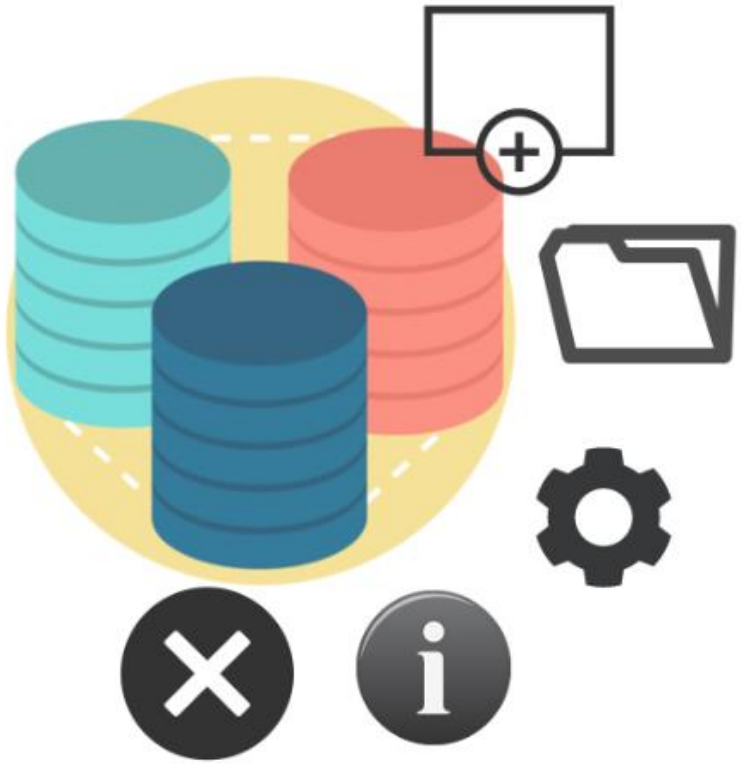
Розроб.	*Кошова А.О.*		18.12	*Діаграмма порівння схем баз даних*	
Перевір.	*Гребеннік І.В.*		18.12		
Н. Контр.	*Гребеннік І.В.*		18.12		
Затверд.	*Гребеннік І.В.*		18.12	*СТ*	*Листів 1*

ER-ДІАГРАМА СЛОВНИКІВ ДЛЯ ПОРІВНЯННЯ ІМЕН ВІДНОСИН І АТРИБУТІВ



Розроб.	*Кошова А.О.*		18.12	*ER-діаграма словників для порівння імен відносин і атрибутів*	
Перевір.	*Гребеннік І.В.*		18.12		
Н. Контр.	*Гребеннік І.В.*		18.12		
Затверд.	*Гребеннік І.В.*		18.12	*СТ*	*Листів 1*

ГОЛОВНА СТОРІНКА ПРОГРАМИ



Розроб.	Кошова А.О.		18.12	*Головна сторінка програми*	
Перевір.	Гребеннік І.В.		18.12		
Н. Контр.	Гребеннік І.В.		18.12		
Затверд.	Гребеннік І.В.		18.12	*СТ*	Листів 1

Додаток Б
Текст програми

ГЮИК. 50 8140.001 – 01 12 01

(позначення документу)

ЗАТВЕРДЖЕНО
ГЮИК.50 8140.001 – 01 12 01 – ЛУ

Дослідження процесів міграції баз даних в корпоративних системах

Текст програми

ГЮИК.50 8140.001 – 01 12 01 – ЛУ

ЛИСТІВ 10

2019 р.

Міністерство освіти і науки України
Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»
керівник атестаційноїроботи
проф. Гребеннік І.В.

Дослідження процесів міграції баз даних в корпоративних системах

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК.ГЮИК.503200.012 – 01 12 01 – ЛУ

РОЗРОБИЛА:
ст. гр. ИТПм-18-1
Кошова А.О.

2019 р.

```
/**
 * Abstract class for loading database parts.
 *
 */
@Component
public abstract class AbstractLoader implements Loader {

    protected Connection connection = null;

    private List<AbstractLoader> childrenLoaders;

    public AbstractLoader() {
        childrenLoaders = new ArrayList<>();
    }

    public AbstractLoader(Connection connection) {
        this.connection = connection;
        childrenLoaders = new ArrayList<>();
    }

    /**
     * This method establish connection for each loader.
     *
     * @param connection
     */
    @Override
    public final void setConnection(Connection connection) {
        this.connection = connection;
    }


    public final TreeNode loadNode(List<String> parameters) throws SQLException {
        TreeNode node = lazyLoad(parameters);
        loadChildren(node, parameters);
        return node;
    }


    /**
     * The method is an implementation of lazy loading for an item.
     *
     * @param parameters
     * @return node
     * @throws SQLException
     */
    public abstract TreeNode lazyLoad(List<String> parameters) throws SQLException;

    public abstract void declareParent(TreeNode<XmlNodeModel> parent, List<String> parameters) throws SQLException;

    @SuppressWarnings("unchecked")
    private final void loadChildren(TreeNode node, List<String> parameters) throws SQLException {
        for(AbstractLoader abstractLoader: childrenLoaders) {
            abstractLoader.declareParent(node,parameters);
        }
    }
```

```
/**
 * Class for loading full metadata from DB.
 */
public class TreeNodeLoader {

   private static TreeNodeLoader instance;

   private static ParamLoader paramLoader = new ParamLoader();

   private TreeNodeLoader() {
   }

   public static synchronized TreeNodeLoader getInstance() {
      if (instance == null) {
         synchronized (TreeNodeLoader.class) {
            if (instance == null) {
               instance = new TreeNodeLoader();
            }
         }
      }
      return instance;
   }

   /**
    * Constructor for the formation of the table.
    *
    * @param parentName
    * @param name
    * @param schemaName
    * @param tables
    * @param loader
    * @throws SQLException
    */
   private static void tableConstructor(String parentName, String name, String schemaName,
List<TreeNode<XmlNodeModel>> tables, Loader loader, Connection connection) throws SQLException {
      TreeNode<XmlNodeModel> columnParent = new TreeNode<>(new XmlNodeModel(parentName));
      loader.setConnection(connection);
      List<TreeNode<XmlNodeModel>> column = loader.fullLoad(schemaName);
      List<TreeNode<XmlNodeModel>> columnTable = new ArrayList<>();
      for (int i = 0; i < tables.size(); i++) {
         for (int j = 0; j < column.size(); j++) {
            if
(tables.get(i).getData().getAttributes().containsValue(column.get(j).getData().getAttributes().get("TABLE_NAME"))) {
               columnParent = new TreeNode<>(new XmlNodeModel(parentName));
               columnTable.add(column.get(j));
            }
         }
         if (columnParent != null) {
            columnParent.addChildren(columnTable);
            tables.get(i).addChild(columnParent);
            columnParent = null;
         }
      }
   }
```

```
    /**
     * Constructor for the formation of the views,procedures and functions.
     *
     * @param name
     * @param loader
     * @param schemaName
     * @return
     * @throws SQLException
     */
    private static TreeNode<XmlNodeModel> xmlConstructor(String name, Loader loader, String schemaName, Connection
connection) throws SQLException {
        TreeNode<XmlNodeModel> parent = new TreeNode<>(new XmlNodeModel(name));
        loader.setConnection(connection);
        List<TreeNode<XmlNodeModel>> child = loader.fullLoad(schemaName);
        for (TreeNode treeNode : child) {
            parent.addChild(treeNode);
        }
        paramConstructor(parent, schemaName, connection);
        return parent;
    }

    /**
     * Constructor for the formation of the parameters.
     *
     * @param parent
     * @param schemaName
     * @throws SQLException
     */
    private static void paramConstructor(TreeNode<XmlNodeModel> parent, String schemaName, Connection connection)
throws SQLException {
        paramLoader.setConnection(connection);
        List<TreeNode<XmlNodeModel>> child = paramLoader.fullLoad(schemaName);
        List<TreeNode<XmlNodeModel>> parents = parent.getChildren();
        for (int i = 0; i < parents.size(); i++) {
            for (int j = 0; j < child.size(); j++) {
                if
(parents.get(i).getData().getAttributes().containsValue(child.get(j).getData().getAttributes().get("PARAMETER_NAME"))) {
                    parents.get(i).addChild(child.get(j));
                }
            }
        }
    }

    public TreeNode fullLoadTree(String schemaName, Connection connection) throws SQLException {
        ApplicationContext context =
            new AnnotationConfigApplicationContext("ua.dbbest.koshova.db.loader");
        List<String> param = new ArrayList<>();
        param.add(schemaName);
        SchemaLoader schemaLoader = (SchemaLoader) context.getBean(DBConst.DBEntity.SCHEMA);
        schemaLoader.setConnection(connection);
        TreeNode root = schemaLoader.lazyLoad(param);
        TreeNode<XmlNodeModel> t = new TreeNode<>(new XmlNodeModel(DBConst.Names.TABLES));
        TableLoader tableLoader = (TableLoader) context.getBean(DBConst.DBEntity.TABLE);
        tableLoader.setConnection(connection);
```

```
        List<TreeNode<XmlNodeModel>> tables = tableLoader.fullLoad(schemaName);
        for (TreeNode table : tables) {
            t.addChild(table);
        }
        root.addChild(t);
        tableConstructor(DBConst.Names.COLUMNS, DBConst.Names.COLUMNS, schemaName, tables, (ColumnLoader)
context.getBean(DBConst.DBEntity.COLUMN), connection);
        tableConstructor(DBConst.Names.INDEXES, DBConst.Names.INDEXES, schemaName, tables, (IndexLoader)
context.getBean(DBConst.DBEntity.INDEX), connection);
        tableConstructor(DBConst.Names.PRIMARY_KEYS, DBConst.Names.PRIMARY_KEYS, schemaName, tables,
(PKLoader) context.getBean(DBConst.DBEntity.PRIMARY_KEY), connection);
        tableConstructor(DBConst.Names.FOREIGN_KEYS, DBConst.Names.FOREIGN_KEYS, schemaName, tables,
(FKLoader) context.getBean(DBConst.DBEntity.FOREIGN_KEY), connection);
        tableConstructor(DBConst.Names.TRIGGERS, DBConst.Names.TRIGGERS, schemaName, tables, (TriggerLoader)
context.getBean(DBConst.DBEntity.TRIGGER), connection);

        TreeNode<XmlNodeModel> v = xmlConstructor(DBConst.Names.VIEWS, (ViewLoader)
context.getBean(DBConst.DBEntity.VIEW), schemaName, connection);
        TreeNode<XmlNodeModel> p = xmlConstructor(DBConst.Names.PROCEDURES, (ProcedureLoader)
context.getBean(DBConst.DBEntity.PROCEDURE), schemaName, connection);
        TreeNode<XmlNodeModel> f = xmlConstructor(DBConst.Names.FUNCTIONS, (FunctionLoader)
context.getBean(DBConst.DBEntity.FUNCTION), schemaName, connection);

        root.addChild(v);
        root.addChild(p);
        root.addChild(f);
        return root;
    }


public class ConnectionManager {
    private static final Logger LOGGER = Logger.getLogger(ConnectionManager.class);

    /**
     * Method to close the connection with the database
     *
     * @param connection
     */
    public static void closeConnection(Connection connection) {
        try {
            connection.close();
        } catch (SQLException e) {
            LOGGER.error("Could not close connection", e);
        }
    }

    /**
     * Method to establish a connection with the database.
     *
     * @return connection.
     */
    public static Connection getConnection(String url, String user, String pass) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection connection = DriverManager.getConnection(url,user,pass);
```

```
            return connection;
        } catch (SQLException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
}

import React from 'react';
import CheckboxTree from 'react-checkbox-tree';
import 'react-checkbox-tree/lib/react-checkbox-tree.css';


class Schema extends React.Component {


    constructor(props) {
        super(props);

        this.onCheck = this.onCheck.bind(this);
        this.onClick = this.onClick.bind(this);
        this.onExpand = this.onExpand.bind(this);
        this.onFilterChange = this.onFilterChange.bind(this);
        this.filterTree = this.filterTree.bind(this);
        this.filterNodes = this.filterNodes.bind(this);
        const replacedTree = this.replaceTree(props.tree);
        console.log(replacedTree);

        var nodes = [];
        nodes.push(replacedTree);

        this.state = {
            uiTree: replacedTree,
            checked: [],
            expanded: [],
            nodesFiltered: nodes,nodes,
            clicked: {},
            filterText: ''
        };
        console.log(this.props.currentNode);
        console.log(this.state.uiTree);
    }

    currentNode = this.props.currentNode;

    replaceTree(treeNode) {
        const replacedTreeNode = {};
        let currPath = "";
        this.recursiveReplaced(replacedTreeNode, treeNode, currPath);
        return replacedTreeNode;
        console.log(treeNode);
    }

    recursiveReplaced(replacedTreeNode, tree, currPath) {
        if (!tree) {
            return;
```

```
    }

    let name =  this.findName(tree.data.attributes, tree.data.name);
    let replacedCurrPath = currPath + (currPath.length > 0 ? "." : "") + name;
    replacedTreeNode.data = tree.data;
    replacedTreeNode.value = replacedCurrPath;
    replacedTreeNode.label = name;
    replacedTreeNode.children = tree.children.map(child => {
        const replacedChild = {};
        this.recursiveReplaced(replacedChild, child, replacedCurrPath);
        return replacedChild;
    });
}

onClick(clicked) {
    var data = this.searchNode(clicked.value).data;
    console.log(data);
    const test = this.currentNode(data);
    if (data.name !== "schema") {
        this.props.updateParentState(data);
    }
    console.log(test);
    this.setState({clicked});
}

onCheck(checked) {
    this.setState({checked});
}

onExpand(expanded) {
    this.setState({expanded});
}


updateParentState(data) {
    console.log(data);
    this.props.updateParentState(data);
}

onFilterChange(e) {
    this.setState({filterText: e.target.value}, this.filterTree);
}

filterTree() {
    // Reset nodes back to unfiltered state
    if (!this.state.filterText) {
        this.setState(prevState => ({
            nodesFiltered: prevState.nodes,
        }));

        return;
    }

    const nodesFiltered = prevState => (
        {nodesFiltered: prevState.nodes.reduce(this.filterNodes, [])}
```

```
    );

    this.setState(nodesFiltered);
}

filterNodes(filtered, node) {
    const {filterText} = this.state;
    const children = (node.children || []).reduce(this.filterNodes, []);

    if (
        // Node's label matches the search string
    node.label.toLocaleLowerCase().indexOf(filterText.toLocaleLowerCase()) > -1 ||
    // Or a children has a matching node
    children.length
    ) {
        filtered.push({...node, children});
    }

    return filtered;
}

render() {
    const {checked, expanded, clicked, filterText,nodesFiltered} = this.state;
    const notClickedText = '(none)';

    console.log(nodesFiltered);

    return (
        <div className="filter-container">
            <input
                className="filter-text"
                placeholder="Search..."
                type="text"
                value={filterText}
                onChange={this.onFilterChange}
            />
            <CheckboxTree
                checked={checked}
                expanded={expanded}
                nodes={nodesFiltered}
                expandOnClick
                onCheck={this.onCheck}
                onClick={this.onClick}
                onExpand={this.onExpand}
            />
            <div className="clickable-labels-info">
                <strong>Clicked Node</strong>: {clicked.value || notClickedText}
            </div>
        </div>
    );
}

findName(attributes, name) {
    switch (name) {
        case "table" :
```

```
            return attributes["TABLE_NAME"];
          case "column" :
            return attributes["COLUMN_NAME"];
          case "schema" :
            return attributes["SCHEMA_NAME"];
          case "view" :
            return attributes["TABLE_NAME"];
          case "function" :
            return attributes["SPECIFIC_NAME"];
          case "procedure" :
            return attributes["SPECIFIC_NAME"];
          case "parameter" :
            return attributes["PARAMETER_NAME"];
          case "primary_key" :
            return attributes["COLUMN_NAME"];
          case "foreign_key" :
            return attributes["CONSTRAINT_NAME"];
          case "trigger" :
            return attributes["TRIGGER_NAME"];
          case "index":
            return attributes["INDEX_NAME"];
        }
        return name;
    }

    searchNode(value) {
        const tree = this.state.uiTree;
        this.updateParentState(this.state.uiTree);
        console.log(value);
        let node = this.recSearch(tree, value);
        return node;
    }

import React from 'react';
import "./style.css";
import {Modal, Button} from 'react-bootstrap';

function Modals() {
    const [show, setShow] = React.useState(false);

    const handleClose = () => setShow(false);
    const handleShow = () => setShow(true);

    return (
        <>
        <Button variant="btn btn-info" onClick={handleShow}>
            Help
        </Button>

        <Modal show={show} onHide={handleClose}>
            <Modal.Header closeButton>
                <Modal.Title>Choose an action:</Modal.Title>
            </Modal.Header>
            <Modal.Body>
                <p> Choose an action:</p>
```

```
            <p>1) Upload file</p>
            <p>2) Open the file</p>
            <p>3) Settings</p>
            <p>4) Info</p>
            <p>5) Exit</p>
          </Modal.Body>
      </Modal>
      </>
    );
}

class Home2 extends React.Component {

   render() {
      return (
         <div>
            <Modals/>
            <div className="container">
               <div className="text-center"><h1>Welcome to application</h1></div>
               <div className="navbar"><img alt={"img"} src="assets/base2.png"/>
                  <ul className="menu">
                     <li><a href='./file'><img src="assets/create2.png" alt="create"/></a></li>
                     <li><a href='./Connection'><img src="assets/open2.png" alt="open"/></a></li>
                     <li><a href='./setting'><img src="assets/setting2.png" alt="setting"/></a></li>
                     <li><a href='./About'><img src="assets/info2.png" alt="info"/></a></li>
                     <li><a href="./close"><img alt="close" src="assets/close2.png"/></a></li>
                  </ul>
               </div>
            </div>
         </div>
      )
   }
}

export default Home2;
   recSearch(tree, value) {
      if (tree.value === value) {
         return tree;
      } else {
         for (let i = 0; i < tree.children.length; i++) {
            const child = tree.children[i];
            const find = this.recSearch(child, value);
            if (find) {
               return find;
            }
         }
      }
      console.log()
   }
}

export default Schema;
     var v = evaluate(new_seq);
     if(v < bestv) {bestv = v, seq = new_seq; };
   }
```

```
public class ProcedurePrinter extends Printer {

  private static final String SQL_CREATE_SP = "DELIMITER $$\n"
     + "CREATE PROCEDURE %s%s\n%s%s%s%s%s"
     + "%s$$\n"
     + "DELIMITER ;";


  @Override
  public String printElement(TreeNode<XmlNodeModel> node) {
     String language = node.getData().getAttributes().get(DBConst.AttributeName.EXTERNAL_LANGUAGE);
     String comment = node.getData().getAttributes().get(DBConst.AttributeName.ROUTINE_COMMENT);
     String determine = "";
     String bodyType = node.getData().getAttributes().get(DBConst.AttributeName.SQL_DATA_ACCESS);
     String security = node.getData().getAttributes().get(DBConst.AttributeName.SECURITY_TYPE);
     if ("NO".equals(node.getData().getAttributes().get(DBConst.AttributeName.IS_DETERMINISTIC))) {
        determine = "NOT DETERMINISTIC";
     } else if ("YES".equals(node.getData().getAttributes().get(DBConst.AttributeName.IS_DETERMINISTIC))) {
        determine = "DETERMINISTIC";
     }
     return String.format(SQL_CREATE_SP, node.getData().getAttributes().get("name"),
        printParams(node),
        determine + "\n",
        comment == null ? "" : comment + "\n",
        language == null ? "" : "LANGUAGE " + language + "\n",
        bodyType == null ? "" : bodyType + "\n",
        security == null ? "" : "SQL SECURITY " + security + "\n",
        node.getData().getAttributes().get(DBConst.AttributeName.ROUTINE_DEFINITION));
  }

  private String printParams(TreeNode<XmlNodeModel> node) {
     StringBuilder sb = new StringBuilder();
     List<TreeNode<XmlNodeModel>> p = node.getChildren();
     if (p.size() == 0) {
        return "";
     } else {
        sb.append("(");
        node.getChildren().get(0).getChildren().forEach((child) -> sb.append(child.getData().getAttributes()
           .get(DBConst.AttributeName.PARAMETER_MODE))
           .append(" ").append(child.getData().getAttributes().get(DBConst.AttributeName.NAME))
           .append(" ").append(child.getData().getAttributes().get(DBConst.AttributeName.DTD_IDENTIFIER)).append(","));
        sb.deleteCharAt(sb.length() - 1);
        sb.append(")");
     }
     return sb.toString();
  }
}
```

Додаток Г
«Специфікація»

ГЮИК. 50 8140.001
(позначення документу)

Код	Назва	Примітки
	Документація	
ГЮИК. 50 8140.001	Текст програми	Додаток Б

Зм.	Лист	№ докум	Підп.	Дата	ГЮИК.503200.012			
Розроб.		Кошова		18.12	*Дослідження процесів міграції баз даних в корпоративних системах*		Лист	Листів
Перевірив.		Гребеннік І.В		18.12			*1*	*1*
Н. Контр.		Гребеннік І.В		18.12	Специфікація	*ХНУРЕ Кафедра СТ*		
Затверд.		Гребеннік І.В.		18.12				

Додаток Д
«Відомість атестаційної роботи»

ГЮИК. 50 8140.001 Д3

(позначення документу)

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1.	ГЮИК. 50 8140.001 ПЗ	Пояснювальна записка	69с.
2.	ГЮИК. 50 8140.001 - 01 12 01 М	«Текст програми»	11с.
		Графічні документи	
5.			1с.
6.	ГЮИК. 50 8140.001 С10		1с.
7.	ГЮИК. 50 8140.001 С10		1с.
8.			1с.
9.			1с.
10.			1с.
11.			1с.

					ГЮИК. 50 8140.001 ДЗ		
					Дослідження процесів міграції баз даних в корпоративних системах	*Лит.*	*Маса* / *Масштаб*
Ізм.	*Лист*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		*Кошова А.О.*		*18.12*			
Перевір.		*Гребеннік І. В.*		*18.12*		*Лист 1*	*Листів 1*
Т. Контр.							
Реценз.					Відомість атестаційної роботи	*ХНУРЕ Кафедра СТ*	
Н. Контр.		*Гребеннік І. В.*		*18.12*			
Затверд.		*Гребеннік І. В.*		*18.12*			