CS306 Project Phase III Report

Web Integrations of SQL and NoSQL Databases
Military Intelligence Database System

Buğra Aydın 30618
Toprak Aktepe 30755
Görkem Subaşı 31023
Bora Çelikörs 30900
Taylan Irak 30702

May 28, 2025

Contents

1 Introduction

This project extends our Military Intelligence Database system into a fully functional web application using PHP and XAMPP. The system integrates both MySQL (for core database operations) and MongoDB (for support ticket management), providing user and administrator interfaces for database interaction.

The application consists of:

- **User Interface**: Accessible via `localhost/user` for trigger/procedure testing and support ticket creation
- **Admin Interface**: Accessible via `localhost/admin` for support ticket management
- **Database Integration**: MySQL for core operations, MongoDB for support tickets

2 System Design and Features

2.1 Project Structure

The project is hosted locally using the XAMPP server with the following directory structure:

```
htdocs/
 user/
    index.php (User homepage)
    support.php (Ticket creation)
    view_tickets.php (Ticket viewing)
    trigger1.php, procedure1.php (Feature interfaces)
 admin/
    index.php (Admin dashboard)
    ticket_detail.php (Ticket management)
 config/
     mysql.php (MySQL connection)
     mongodb.php (MongoDB connection)
```

2.2 Technology Stack

- **Web Server**: XAMPP (Apache)
- **Backend**: PHP 8.2.12
- **Databases**: MySQL (core data), MongoDB (support tickets)
- **Frontend**: HTML, CSS, JavaScript

3 Trigger Implementations

3.1 Trigger 1: Drone Operator Assignment Validation

Responsible Member: Buğra Aydın

Description: Validates drone operator assignments and logs all changes to the DroneStatus table.

SQL Script:

```
CREATE TRIGGER operator_assignment_validation
AFTER UPDATE ON Drones
FOR EACH ROW
BEGIN
    IF NEW.op_id != OLD.op_id THEN
        INSERT INTO DroneStatus (drone_id, old_operator_id,
           new_operator_id,
                                 status_date, status_message)
        VALUES (NEW.drone_id, OLD.op_id, NEW.op_id, NOW(),
                CONCAT('Operator assignment changed from ', OLD.op_id,
                       ' to ', NEW.op_id));
    END IF;
END
```

Web Interface Cases:

- **Case 1**: "Assign Operator 1 to Drone 1" - Tests valid operator assignment
- **Case 2**: "Assign Operator 2 to Drone 1" - Tests operator reassignment
- **Case 3**: "Assign Operator 3 to Drone 2" - Tests assignment validation

3.2 Trigger 2: Intelligence Report Update Logging

Responsible Member: Toprak Aktepe

Description: Copies the old and new content of an intelligence report into the report_audit table every time the report is edited.

SQL Script:

```
CREATE TRIGGER intelligence_report_audit
AFTER INSERT ON Intelligence_Reports
FOR EACH ROW
BEGIN
    INSERT INTO Report_Audit (report_id, title, classification_level,
                              created_date, agent_id)
    VALUES (NEW.report_id, NEW.title, NEW.classification_level,
            NEW.date_created, NEW.agent_id);
END
```

Web Interface Cases:

- **Case 1**: "Create Top Secret Report" - Creates report with Top Secret classification
- **Case 2**: "Create Classified Report" - Creates report with Classified classification
- **Case 3**: "Create Unclassified Report" - Creates report with Unclassified classification

3.3 Trigger 3: Vehicle Status Management

Responsible Member: Görkem Subaşı

Description: Switches a vehicle's status between Available, Reserved and Maintenance automatically when reservations or returns occur.

SQL Script:

```
CREATE TRIGGER vehicle_status_update
AFTER UPDATE ON Vehicles
FOR EACH ROW
BEGIN
    IF NEW.operational_status != OLD.operational_status THEN
        INSERT INTO VehicleStatusLog (vehicle_id, old_status,
           new_status,
                                      change_date, change_reason)
        VALUES (NEW.vehicle_id, OLD.operational_status, NEW.
           operational_status,
             NOW(), CONCAT('Status changed from ', OLD.
                operational_status,
                          ' to ', NEW.operational_status));
    END IF;
END
```

Web Interface Cases:

- **Case 1**: "Active to Maintenance" - Changes vehicle status from Active to Maintenance
- **Case 2**: "Maintenance to Repair" - Escalates status from Maintenance to Repair
- **Case 3**: "Repair to Active" - Returns vehicle from Repair back to Active

3.4 Trigger 4: Supply Inventory Management

Responsible Member: Bora Çelikörs

Description: Decreases stock on issue, restores it on cancellation, and blocks transactions that would drive inventory below zero.

SQL Script:

```
CREATE TRIGGER supply_inventory_management
AFTER UPDATE ON Supply
FOR EACH ROW
BEGIN
    IF NEW.quantity != OLD.quantity THEN
        INSERT INTO Supply_Audit (supply_id, old_quantity, new_quantity
           ,
                                  audit_date, audit_message)
        VALUES (NEW.supply_id, OLD.quantity, NEW.quantity, NOW(),
                CONCAT('Quantity changed from ', OLD.quantity,
                       ' to ', NEW.quantity));

        IF NEW.quantity <= 50 THEN
            INSERT INTO Supply_Audit (supply_id, old_quantity,
               new_quantity,
                                      audit_date, audit_message)
            VALUES (NEW.supply_id, OLD.quantity, NEW.quantity, NOW(),
                    'LOW STOCK ALERT: Quantity below threshold!');
```

```
        END IF;
    END IF;
END
```

Web Interface Cases:

- **Case 1**: "Normal Stock Reduction" - Updates supply quantity and logs change
- **Case 2**: "Low Stock Alert" - Reduces quantity below threshold, triggering alert
- **Case 3**: "Stock Increase" - Increases supply quantity and logs update

3.5 Trigger 5: Agent Activity Logging

Responsible Member: Taylan Irak

Description: Writes a timestamped entry to agent_activity_log whenever an agent creates, updates or deletes mission-critical data.

SQL Script:

```
CREATE TRIGGER agent_activity_logging
AFTER UPDATE ON Agents
FOR EACH ROW
BEGIN
    IF NEW.`rank` != OLD.`rank` THEN
        INSERT INTO Agent_Activity_Log (agent_id, old_rank, new_rank,
                                        activity_date, activity_type,
                                        activity_description)
        VALUES (NEW.agent_id, OLD.`rank`, NEW.`rank`, NOW(), '
           RANK_CHANGE',
                CONCAT('Agent rank changed from ', OLD.`rank`,
                       ' to ', NEW.`rank`));
    END IF;
END;

CREATE TRIGGER agent_report_logging
AFTER INSERT ON Intelligence_Reports
FOR EACH ROW
BEGIN
    INSERT INTO Agent_Activity_Log (agent_id, old_rank, new_rank,
                                    activity_date, activity_type,
                                    activity_description)
    VALUES (NEW.agent_id, '', '', NOW(), 'REPORT_CREATED',
            CONCAT('Agent created intelligence report: ', NEW.title));
END
```

Web Interface Cases:

- **Case 1**: "Agent Promotion" - Promotes agent and logs rank change
- **Case 2**: "Create Intelligence Report" - Creates report and logs activity
- **Case 3**: "Multiple Activity Simulation" - Combines promotion and report creation

4 Stored Procedure Implementations

4.1 Procedure 1: AssignOperatorToDrone

Responsible Member: Buğra Aydın

Description: Creates an operator-drone assignment after confirming eligibility and marks the drone as Busy.

SQL Script:

```
DELIMITER //
CREATE PROCEDURE AssignOperatorToDrone(
    IN p_operator_id INT,
    IN p_drone_id INT,
    IN p_rank VARCHAR(50)
)
BEGIN
    DECLARE v_operator_exists INT;
    DECLARE v_drone_exists INT;

    SELECT COUNT(*) INTO v_operator_exists
    FROM Operator WHERE op_id = p_operator_id;

    IF v_operator_exists = 0 THEN
        INSERT INTO Operator (op_id, `rank`)
        VALUES (p_operator_id, p_rank);
    ELSE
        UPDATE Operator SET `rank` = p_rank
        WHERE op_id = p_operator_id;
    END IF;

    SELECT COUNT(*) INTO v_drone_exists
    FROM Drones WHERE drone_id = p_drone_id;
    IF v_drone_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid drone ID. Drone does not exist.';
    END IF;

    UPDATE Drones SET op_id = p_operator_id
    WHERE drone_id = p_drone_id;

    SELECT CONCAT('Operator ', p_operator_id, ' assigned to drone ',
                  p_drone_id) AS result;
END//
DELIMITER ;
```

Input Parameters:

- **Operator ID** (INT): Input box for operator identifier
- **Drone ID** (INT): Input box for drone identifier
- **Rank** (VARCHAR): Input box for operator rank

4.2 Procedure 2: GetAgentReports

Responsible Member: Toprak Aktepe

Description: List all reports written by an agent.
SQL Script:

```
DELIMITER //
CREATE PROCEDURE GetAgentReports(
    IN p_agent_id INT
)
BEGIN
    DECLARE agent_name VARCHAR(100);

    SELECT name INTO agent_name FROM Agents WHERE agent_id = p_agent_id
        ;

    SELECT CONCAT('Agent: ', agent_name, ' (ID: ', p_agent_id, ')')
           AS agent_info;

    SELECT report_id, date_created, title, classification_level
    FROM Intelligence_Reports
    WHERE agent_id = p_agent_id
    ORDER BY date_created DESC;

    SELECT COUNT(*) AS total_reports,
           MAX(date_created) AS most_recent_report,
           MIN(date_created) AS oldest_report
    FROM Intelligence_Reports
    WHERE agent_id = p_agent_id;
END//
DELIMITER ;
```

Input Parameters:

- **Agent ID** (INT): Input box for agent identifier

4.3 Procedure 3: GenerateIntelligenceReport

Responsible Member: Taylan Irak
Description: Generate consolidated intelligence report.
SQL Script:

```
DELIMITER //
CREATE PROCEDURE GenerateIntelligenceReport(
    IN p_agent_id INT,
    IN p_title VARCHAR(200),
    IN p_content TEXT,
    IN p_classification_level VARCHAR(50)
)
BEGIN
    DECLARE v_agent_exists INT;
    DECLARE v_report_id INT;

    SELECT COUNT(*) INTO v_agent_exists
    FROM Agents WHERE agent_id = p_agent_id;
    IF v_agent_exists = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Invalid agent ID. Agent does not exist.';
    END IF;

    IF p_title IS NULL OR TRIM(p_title) = '' THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Report title cannot be empty.';
    END IF;

    SELECT COALESCE(MAX(report_id), 0) + 1 INTO v_report_id
    FROM Intelligence_Reports;

    INSERT INTO Intelligence_Reports (report_id, date_created, title,
                                      content, classification_level,
                                         agent_id)
    VALUES (v_report_id, NOW(), p_title, p_content,
            p_classification_level, p_agent_id);

    SELECT CONCAT('Intelligence report created successfully. Report ID:
        ',
                  v_report_id) AS message;
END//
DELIMITER ;
```

Input Parameters:

- **Agent ID** (INT): Input box for agent identifier
- **Title** (VARCHAR): Input box for report title
- **Content** (TEXT): Text area for report content
- **Classification Level** (VARCHAR): Input box for security classification

4.4 Procedure 4: ReserveVehicle

Responsible Member: Görkem Subaşı
Description: Update vehicle status via reservation.
SQL Script:

```
DELIMITER //
CREATE PROCEDURE ReserveVehicle(
    IN p_base_id INT,
    IN p_vehicle_type VARCHAR(50)
)
BEGIN
    DECLARE v_vehicle_id INT;

    SELECT vehicle_id INTO v_vehicle_id
    FROM vehicles
    WHERE base_id = p_base_id
      AND type = p_vehicle_type
      AND operational_status = 'Active'
    LIMIT 1;

    IF v_vehicle_id IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No available vehicle found for reservation.
            ';
    ELSE
        UPDATE vehicles SET operational_status = 'Reserved'
        WHERE vehicle_id = v_vehicle_id;
```

```
22
23          SELECT CONCAT('Vehicle ', v_vehicle_id, ' reserved successfully
                .')
24                  AS message;
25      END IF;
26  END//
27  DELIMITER ;
```

Input Parameters:

- **Base ID** (INT): Input box for military base identifier
- **Vehicle Type** (VARCHAR): Input box for vehicle type

4.5 Procedure 5: OrderSupply

Responsible Member: Bora Çelikörs
Description: Create supply order and update quantities.
SQL Script:

```
1  DELIMITER //
2  CREATE PROCEDURE OrderSupply(
3      IN p_supply_id INT,
4      IN p_order_quantity INT
5  )
6  BEGIN
7      DECLARE v_current_quantity INT;
8
9      SELECT quantity INTO v_current_quantity
10     FROM Supply WHERE supply_id = p_supply_id;
11
12     IF v_current_quantity IS NULL THEN
13         SIGNAL SQLSTATE '45000'
14         SET MESSAGE_TEXT = 'Invalid supply ID. Supply does not exist.';
15     ELSE
16         UPDATE Supply SET quantity = quantity + p_order_quantity
17         WHERE supply_id = p_supply_id;
18
19         SELECT CONCAT('Supply ', p_supply_id, ' updated. New quantity:
               ',
20                       v_current_quantity + p_order_quantity) AS message
                             ;
21     END IF;
22 END//
23 DELIMITER ;
```

Input Parameters:

- **Supply ID** (INT): Input box for supply item identifier
- **Order Quantity** (INT): Input box for quantity to order

5 MongoDB Support System Implementation

5.1 Ticket Creation Query

PHP Script:

```php
<?php
function createTicket($username, $message) {
    try {
        $manager = getMongoDBConnection();

        $document = [
            'username' => $username,
            'message' => $message,
            'created_at' => date('Y-m-d H:i:s'),
            'status' => true,
            'comments' => []
        ];

        $bulk = new MongoDB\Driver\BulkWrite;
        $bulk->insert($document);

        $result = $manager->executeBulkWrite(
            MONGODB_DATABASE . '.' . MONGODB_COLLECTION, $bulk);
        return $result->getInsertedCount() > 0;

    } catch (Exception $e) {
        throw new Exception("Failed to create ticket: " . $e->
            getMessage());
    }
}
?>
```

5.2 Ticket Retrieval Query

PHP Script:

```php
<?php
// Get all active tickets for admin
$filter = ['status' => true];
$options = ['sort' => ['created_at' => -1]];
$query = new MongoDB\Driver\Query($filter, $options);
$cursor = $manager->executeQuery(
    MONGODB_DATABASE . '.' . MONGODB_COLLECTION, $query);
$tickets = $cursor->toArray();
?>
```

5.3 Comment Addition Query

PHP Script:

```php
<?php
function addComment($ticketId, $comment, $commenter = 'user') {
    try {
        $manager = getMongoDBConnection();

        $filter = ['_id' => new MongoDB\BSON\ObjectId($ticketId)];
        $update = [
            '$push' => [
                'comments' => [
                    'comment' => $comment,
```

```
                    'commenter' => $commenter,
                    'timestamp' => date('Y-m-d H:i:s')
                ]
            ]
        ];

        $bulk = new MongoDB\Driver\BulkWrite;
        $bulk->update($filter, $update);

        $result = $manager->executeBulkWrite(
            MONGODB_DATABASE . '.' . MONGODB_COLLECTION, $bulk);
        return $result->getModifiedCount() > 0;

    } catch (Exception $e) {
        throw new Exception("Failed to add comment: " . $e->getMessage
            ());
    }
}
?>
```

5.4 Ticket Resolution Query

PHP Script:

```
<?php
function resolveTicket($ticketId) {
    try {
        $manager = getMongoDBConnection();

        $filter = ['_id' => new MongoDB\BSON\ObjectId($ticketId)];
        $update = ['$set' => ['status' => false]];

        $bulk = new MongoDB\Driver\BulkWrite;
        $bulk->update($filter, $update);

        $result = $manager->executeBulkWrite(
            MONGODB_DATABASE . '.' . MONGODB_COLLECTION, $bulk);
        return $result->getModifiedCount() > 0;

    } catch (Exception $e) {
        throw new Exception("Failed to resolve ticket: " . $e->
            getMessage());
    }
}
?>
```

6 User Interface Features

6.1 User Dashboard

The user interface provides:

- Homepage displaying all triggers and procedures with descriptions
- Navigation links to support system

- Responsive design with card-based layout

6.2 Support Ticket System

Features include:

- Ticket creation with username and message fields
- Ticket listing filtered by username
- Comment system for user-admin communication
- Status tracking (active/resolved)

6.3 Admin Interface

Administrative capabilities:

- View all active tickets across all users
- Add comments as admin user
- Mark tickets as resolved
- Ticket detail view with full conversation history

7 Technical Implementation

7.1 Database Connections

- **MySQL**: Using MySQLi extension for relational data
- **MongoDB**: Using MongoDB\Driver\Manager for document operations

7.2 Security Considerations

- Input validation and sanitization
- SQL injection prevention using prepared statements
- XSS protection with htmlspecialchars()

8 Conclusion

The Military Intelligence Database web application successfully integrates both SQL and NoSQL databases, providing a comprehensive platform for database interaction through user-friendly web interfaces. The system demonstrates effective use of triggers, stored procedures, and modern web technologies while maintaining data integrity and security.

Repository: `https://github.com/aydinbugra03/MilitaryIntelligenceDB`