# Technical Documentation

# Introduction

This document presents the design and development of a relational database system for the **SANAD Personal Identification Card Renewal service**, implemented by FutureTech Innovations. The database system is intended to manage citizen information, track service requests, process payments, and provide secure access to authorized employees and administrators.

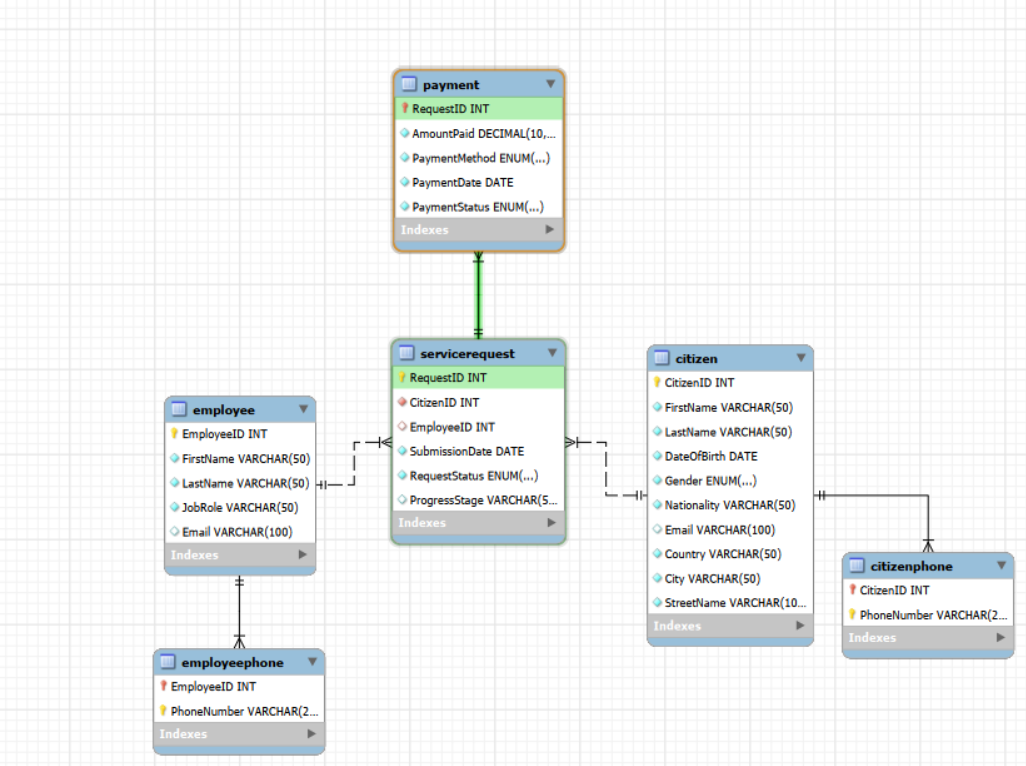
The purpose of this document is to describe the **database structure, components, and design methodology** used to ensure accurate record-keeping, efficient service tracking, and data integrity. It includes detailed explanations of the conceptual, logical, and physical designs, the normalization process, and the relationships between entities.

This system supports multiple user roles, includin g citizens, employees, administrators, and auditors, each with specific privileges and responsibilities. By implementing a structured relational database, the system ensures:

* Secure storage and management of citizen and payment data.
* Accurate tracking of personal ID renewal requests from submission to completion.
* Efficient processing and assignment of requests to employees.
* Generation of reports and audit trails to support transparency and accountability.

The design follows standard database principles to minimize redundancy, enforce data integrity, and optimize performance. It provides a foundation for future expansion and integration with other SANAD services, ensuring that the system can scale and adapt to evolving requirements.

# Physical Schema



# Database Development

The **SANAD Personal Identification Card Renewal** database is designed to manage citizen information, employee processing, service requests, and payment records. The database consists of several interrelated tables that work together to ensure data accuracy, integrity, and efficient tracking of the service workflow.

## Database Overview

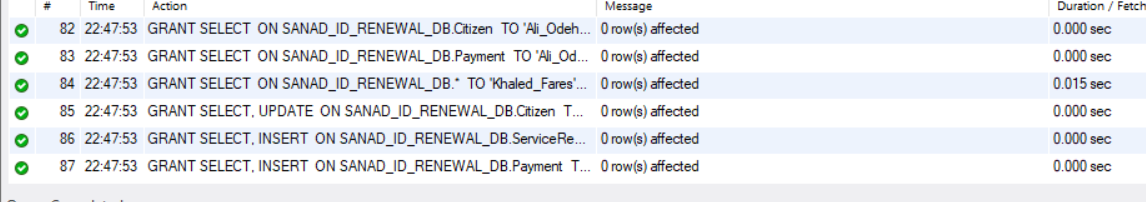
|  |  |  |
| --- | --- | --- |
| **Table** | **Name** | **Description** |
|  | **Citizen** | The table contains the essential attributes required to identify citizens and communicate with them. It’s including the following attributes:(CitizenID, FirstName, LastName, BirthDate, Gender, Nationality, Email, Country, City, StreetName)   * **CitizenID**: This attribute is the primary key of the Citizen table. It uniquely identifies each citizen in the system. It is of type INT, auto incremented, and has a NOT NULL constraint. * **FirstName**: Stores the first name of the citizen. This attribute is of type VARCHAR and has a NOT NULL constraint. * **LastName**: Stores the last name of the citizen. This attribute is of type VARCHAR and has a NOT NULL constraint. * **BirthDate**: Stores the citizen’s date of birth. This attribute is of type DATE and has a NOT NULL constraint. * **Gender**: Stores the gender of the citizen. This attribute is of type VARCHAR and has a NOT NULL constraint. * **Nationality**: Stores the nationality of the citizen. This attribute is of type VARCHAR and has a NOT NULL constraint. * **Email**: Stores the email address of the citizen. This attribute is of type VARCHAR, has a UNIQUE constraint, and is used for communication purposes. * **Country**: Stores the country of residence of the citizen. This attribute is of VARCHAR and allows NULL values. * **City**: Stores the city of residence of the citizen. This attribute is of VARCHAR and allows NULL values. * **StreetName**: Stores the street name of the citizen’s address. This attribute is of type VARCHAR and has a NOT NULL constraint. |
|  | **CitizenPhone** | This table is responsible for storing citizens’ phone numbers. It is used to handle the multivalued phone number attribute for citizens.The table includes the following attributes:(CitizenID, PhoneNumber)   * **CitizenID**: This attribute is a foreign key that references Citizen (CitizenID). It links each phone number to a specific citizen and has a NOT NULL constraint. * **PhoneNumber**: Stores the citizen’s phone number. This attribute is of type VARCHAR and, together with CitizenID, forms the primary key of the table. |
|  | **Employee** | This table stores information about employees responsible for processing and reviewing service requests within the system. The table includes the following attributes: (EmployeeID, FirstName, LastName, JobRole, Email)   * **EmployeeID**: This attribute is the primary key of the Employee table. It uniquely identifies each employee. It is of type INT and has a NOT NULL constraint. * **FirstName**: Stores the employee’s first name. This attribute is of type VARCHAR and has a NOT NULL constraint. * **LastName**: Stores the employee’s last name. This attribute is of type VARCHAR and has a NOT NULL constraint. * **JobRole**: Stores the job role or position of the employee. This attribute is of type VARCHAR and has a NOT NULL constraint. * **Email**: Stores the employee’s email address. This attribute is of type VARCHAR, has a UNIQUE constraint, and is used for official communication. |
|  | **Employee**  **Phone** | This table stores phone numbers for employees, allowing each employee to have more than one contact number. The table includes the following attributes:(EmployeeID, PhoneNumber)   * **EmployeeID**: This attribute is a foreign key that references Employee (EmployeeID). It links phone numbers to employees and has a NOT NULL constraint. * **PhoneNumber**: Stores the employee’s phone number. This attribute is of type VARCHAR and, together with EmployeeID, forms the primary key of the table. |
|  | **Service**  **Request** | This table is responsible for tracking personal identification card renewal requests submitted by citizens. It also records request status and employee assignment. The table includes the following attributes: (RequestID, SubmissionDate, RequestStatus, ProgressStage, CitizenID, EmployeeID)   * **RequestID**: This attribute is the primary key of the ServiceRequest table. It uniquely identifies each service request. It is of type INT and has a NOT NULL constraint. * **SubmissionDate**: Stores the date when the service request was submitted. This attribute is of type DATE and has a NOT NULL constraint. * **RequestStatus**: Stores the current status of the request (e.g., Pending, Under Review, Approved, Rejected). This attribute is of type VARCHAR and has a NOT NULL constraint. * **ProgressStage**: Stores the processing stage of the request. This attribute is of type VARCHAR and has a NOT NULL constraint. * **CitizenID**: This attribute is a foreign key that references Citizen (CitizenID). It identifies the citizen who submitted the request and has a NOT NULL constraint. * **EmployeeID**: This attribute is a foreign key that references Employee (EmployeeID). It identifies the employee responsible for processing the request and has a NOT NULL constraint. |
|  | **Payment** | This table is responsible for storing payment records related to service requests. It ensures that payments are properly linked to their corresponding requests. The table includes the following attributes: (PaymentID, AmountPaid, PaymentMethod, PaymentDate, PaymentStatus, RequestID)   * **PaymentID**: This attribute is the primary key of the Payment table. It uniquely identifies each payment record. It is of type INT and has a NOT NULL constraint. * **AmountPaid**: Stores the amount paid for the service. This attribute is of type DECIMAL and has a NOT NULL constraint. * **PaymentMethod**: Stores the payment method used (e.g., card, e-wallet). This attribute is of type VARCHAR and has a NOT NULL constraint. * **PaymentDate**: Stores the date when the payment was made. This attribute is of type DATE and has a NOT NULL constraint. * **PaymentStatus**: Stores the status of the payment (Completed, Pending, Failed). This attribute is of type VARCHAR and has a NOT NULL constraint. * **RequestID**: This attribute is a foreign key that references ServiceRequest (RequestID). It links each payment to a specific service request and has a NOT NULL constraint. |

|  |  |  |
| --- | --- | --- |
| **View** | **Name** | **Description** |
|  | **View\_ServiceRequests** | This view lists all service requests, combining request details (RequestID, submission date, and status) with citizen information (first and last name) and employee information (first and last name). It helps in quickly checking which employee is handling which citizen request. |
|  | **View\_PaymentSummary** | This view summarizes payments for each service request by combining payment records with citizen information. It shows the total amount paid and the number of payments made per request, allowing for quick financial oversight and tracking. |
|  | **View\_CitizenContacts** | This view provides citizen contact information along with multiple phone numbers. It consolidates email addresses and phone numbers in a single output, making it easier to reach citizens. |
|  | **View\_PendingRequests** | 1. This view lists all pending service requests with citizen details (first and last name, submission date, and request status). It is useful for employees and administrators to monitor requests that require action. |

|  |  |  |
| --- | --- | --- |
| **Procedure** | **Name** | **Description** |
|  | **Get\_Total\_ Payments\_By\_ Citizen** | Calculates the total completed payments for each citizen by summing all payments linked to their service requests. |
|  | **Get\_Pending\_ Requests\_By\_ Employee** | Counts all pending service requests assigned to each employee to help manage workload. |
|  | **Get\_Citizen\_ Request\_Status** | Retrieves all service requests for a specific citizen, including submission date, status, and progress stage. |
|  | **Get\_Payments\_By\_Method** | Provides a summary of completed payments grouped by payment method, including total number and total amount of payments. |

## Security

|  |  |  |  |
| --- | --- | --- | --- |
| **Username** | **Privilege Command** | **Description** | **Screenshot** |
| Hani\_Zaid  (Administrator)  Ali\_Odeh  (Employee) | INSERT, UPDATE, DELETE | Full CRUD privileges on the entire database. Hani\_Zaid can view, add, modify, and delete records in all tables. |  |
| SELECT, UPDATE | Limited privileges: can view all data, update only ServiceRequest table. Cannot add or delete records. |  |
| Khaled\_Fares  (Citizen)  Ahmad\_Haddad  (Service Editor) | SELECT | Read-only access across the entire database. Cannot insert, update, or delete any records. |  |
| SELECT, UPDATE &  SELECT, INSERT | Can view and update Citizen data; view and insert into ServiceRequest and Payment. Cannot delete records. |  |



## User Interface

### Flowchart and Data Movement Diagrams

### Interfaces Development

|  |  |  |  |
| --- | --- | --- | --- |
| **Page ID** | **Title** | **Description** | **Screenshot** |
|  | Database Connection Setup | This page configures the connection between the system and the MySQL database, where the administrator logs in using admin credentials. |  |
|  | Home Page  (Select table page) | From this page the user can see the tables that have access to them and choose which one he want. |  |
|  | Data base table | From this page the user can see the attribute of the chosen table and edit or delete or add on them if he has privilege to do it |  |
|  | Add page | From this page the user can add the values to add new row |  |
|  | Update page | From this page the user can update the values to any row |  |
|  | View page | From this page the user can view the values for any row |  |

# Maintenance

## Database recovery & backups

* **Importance of Data Recovery**

Data recovery is critical for ensuring organizational stability and reliability. It guarantees that valuable information remains accessible even when unexpected problems occur. Organizations rely heavily on digital data for operations, decision-making, and service delivery. Loss of data can arise from:

* Accidental deletions
* Hardware failures
* Software errors
* Cyberattacks
* Natural disasters

Without a proper recovery plan, organizations risk financial loss, operational downtime, reputational damage, and legal implications.

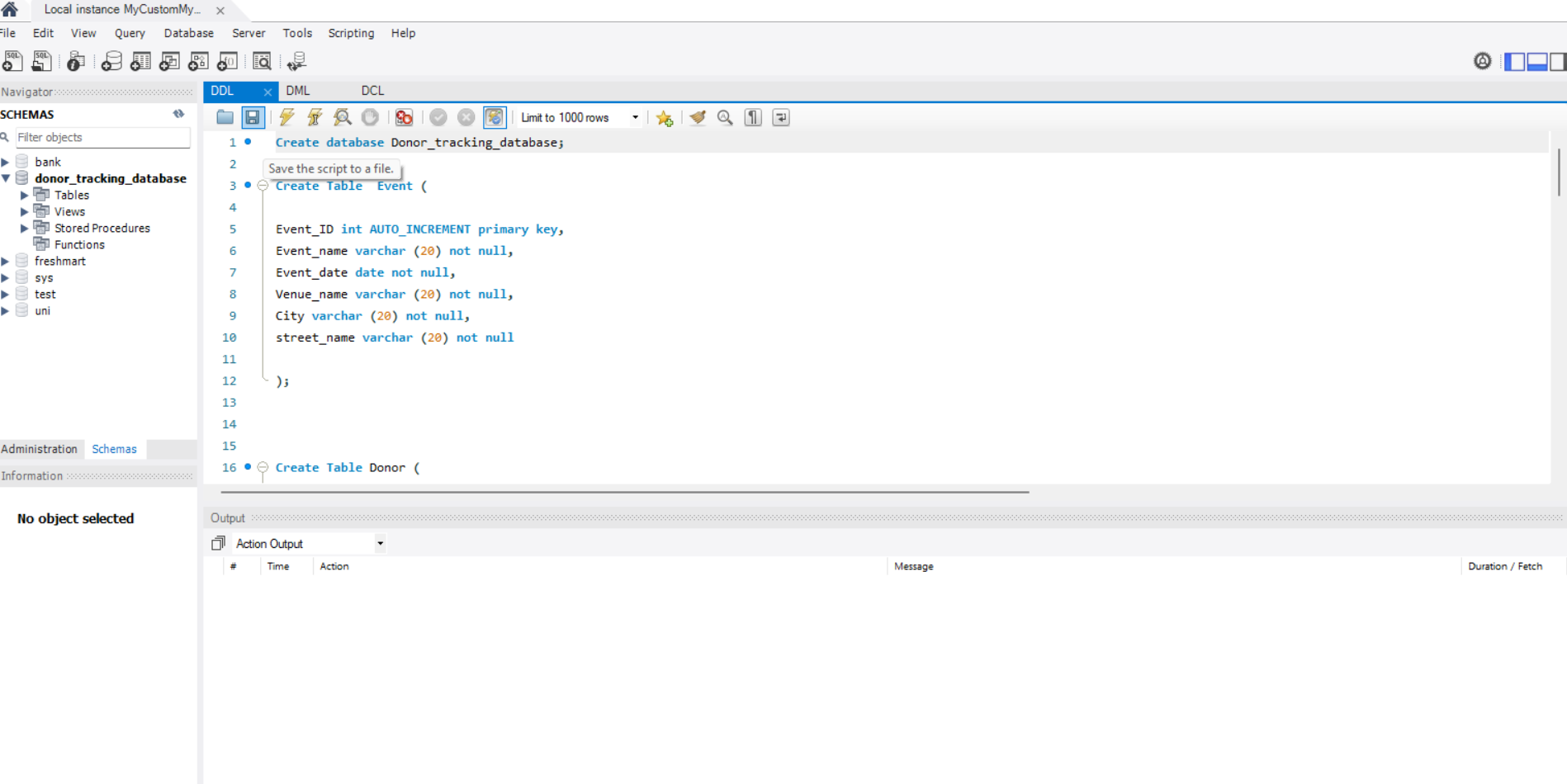
* **Backup and Recovery Criteria**

To ensure effective database recovery, the following criteria should be met:

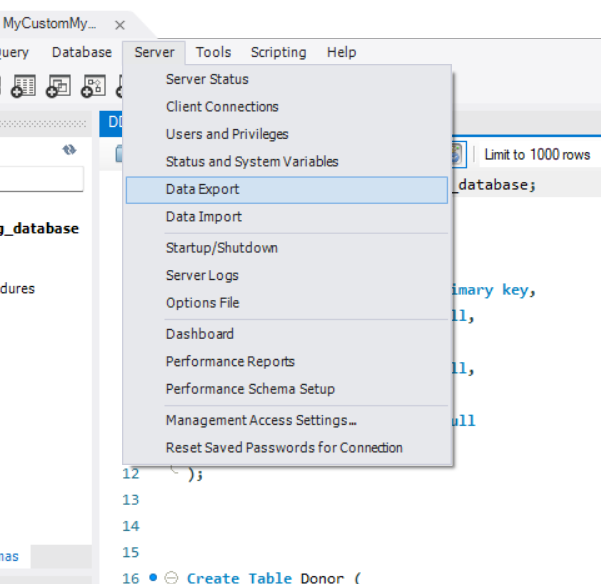
1. **Regular Backups**
   * Full backups of the database should be scheduled at regular intervals (daily, weekly, or monthly, depending on criticality).
   * Incremental or differential backups can be used to save storage space while maintaining data integrity.
2. **Backup Storage and Security**
   * Backups should be stored securely, ideally in multiple locations (on-site and off-site/cloud).
   * Access to backup files must be restricted to authorized personnel to prevent unauthorized tampering or theft.
3. **Testing and Verification**
   * Regularly test backup files to ensure data can be successfully restored.
   * Verify the integrity of backup data to avoid corrupted or incomplete recovery.
4. **Recovery Procedures**
   * Maintain a documented recovery plan outlining steps to restore the database in case of data loss.
   * Recovery procedures should cover partial or full restoration, including the order of restoring dependent data.
5. **Minimizing Downtime**
   * Backup and recovery solutions should aim to minimize operational downtime.
   * Implement automated recovery mechanisms where possible to speed up restoration.
6. **Compliance and Audit**
   * Ensure backup and recovery processes comply with industry standards and legal requirements.
   * Maintain logs of backup activities and recovery operations for auditing purposes.
7. **Preventive Measures**
   * Implement security measures such as encryption, access controls, and monitoring to prevent data loss.
   * Educate staff about best practices for data handling and backup procedures.

**First export:**

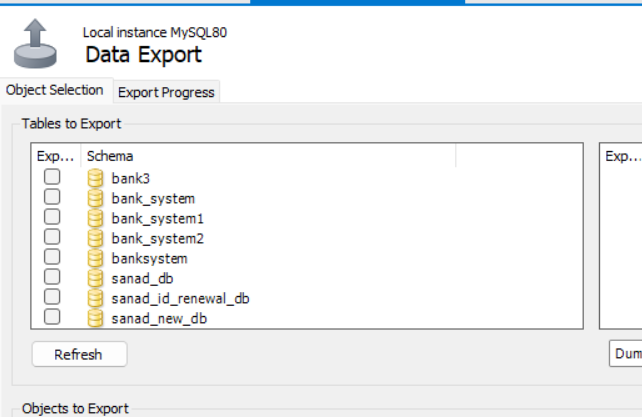
1. First, go to the connection and choose Server.



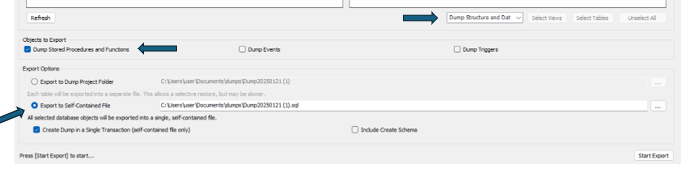
2. After that go to Data export.



3. Then choose the database we want to export



4. Make sure that stored the database with the data and stored the procedure and export them as one file



**Second Data import:**

1. go to server then import database

2.Then this page will appear

3.Choose the database you want to import and choose the schema you want to import in it or create a new one

4.The click on start import and like this we import the database successfully.

## Database maintenance in general

* **Overview**  
  Database maintenance is a critical practice to ensure the reliability, performance, and security of organizational data assets. Proper maintenance activities prevent potential issues such as data loss, system downtime, and compromised data integrity. Proactive monitoring, optimization, and management of databases support operational efficiency and protect sensitive information from unauthorized access.
* **Importance**  
  Maintaining a database ensures that it operates at optimal performance and that data integrity is preserved. Over time, databases may experience fragmentation, accumulation of redundant data, or outdated statistical information, which can degrade performance. Regular maintenance resolves these issues and enables organizations to detect and mitigate security vulnerabilities effectively, safeguarding sensitive data against unauthorized access or breaches.
* **Key Maintenance Activities**

1. **Regular Backups**
   * Essential for recovery in case of hardware failures, accidental deletions, or cyberattacks.
   * Effective backup strategies include full, differential, and incremental backups to provide flexible restoration options. (Seis)
2. **Index Optimization**
   * Index fragmentation can slow query performance.
   * Periodic rebuilding or reorganizing of indexes ensures fast data access and overall database efficiency. (Dimitrov)
3. **Updating Statistics**
   * Databases rely on statistical data for optimal query execution planning.
   * Regular updates ensure the query optimizer has accurate data distribution information, improving performance. (Dimitrov)
4. **Data Integrity Checks**
   * Consistency and integrity checks detect anomalies or corruption in the database.
   * Constraints and validation rules enforce accuracy and reliability of the stored data. (Seis)
5. **Disk Space Management**
   * Monitoring and managing storage prevents performance degradation due to space shortages.
   * Practices include removing unused data and archiving outdated records. (Seis)
6. **Applying Updates and Patches**
   * Keeping the DBMS and associated software updated addresses known vulnerabilities and performance issues.
   * Timely application of vendor-released patches ensures secure and stable operations. (Collins)
7. **Performance Monitoring and Tuning**
   * Continuous monitoring of database metrics identifies bottlenecks or resource-intensive queries.
   * Tuning system parameters and queries ensures optimal use of resources and response times. (Collins)

**Conclusion**  
A structured maintenance plan is essential for the SANAD database system to ensure high performance, data accuracy, and security. Implementing these proactive strategies allows the system to support ongoing operations reliably, respond efficiently to user requests, and maintain a secure environment for sensitive citizen and payment information.

# Testing

## Data Validation

This section should provide a detailed overview of the data validation rules applied within the database to ensure data accuracy, consistency, and integrity.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Type** | **Description** | **screenshot** |
|  | All cases of PK | **Uniqueness:**  - In this test, I will attempt to insert a duplicate primary key into the **Citizen** table by using an existing **CitizenID**.  - The expected outcome is that the system will generate an error and prevent the insertion of a duplicate **CitizenID** primary key. |  |
| **Not Null:**  In this test, I will attempt to insert a **NULL** value into the citizen id .  -The expected output is an error because the primary key can’t be null |  |
|  | All cases of FK | **Non-existed PK:**  In this test, I will attempt to insert a new record into the **EmployeePhone** table using a **non-existent EmployeeID**, which is **19**.  -The EmployeeID in EmployeePhone is a **foreign key** that must reference an existing primary key in the **Employee** table. |  |
| **On update cascade:**  In this test, I will verify the **ON UPDATE CASCADE** constraint between the **Employee** table and the **EmployeePhone** table.  -This is done by **updating the EmployeeID** in the Employee table. | Before :      After: |
| **ON DELETE Restrict**  **Description:**   * In this test, I will verify the **ON DELETE RESTRICT** behavior between the **Employee** table and **EmployeePhone**. * I will attempt to delete an employee who still has a related record in EmployeePhone.   **Expected Output**   * The delete operation should fail with an error because the employee ID is referenced in another table and **RESTRICT prevents deletion**. |  |
| **ON DELETE SET NULL**  **Description:**   * In this test, I will verify the **ON DELETE SET NULL** behavior between the **Employee** table and the **ServiceRequest** table. * I will attempt to delete an employee who is currently assigned to a service request.   **Expected Output:**   * The employee record is deleted successfully. * The **EmployeeID** in the **ServiceRequest** table is automatically set to **NULL**, while the service request record remains unchanged. | Before delete :  After delete: |
|  | Unique | **Description:**   * In this test, I will attempt to insert two records into the **Citizen** table with the same **Email** value. * The **Email** attribute is defined as **UNIQUE**, so duplicate values are not allowed.   **Expected Output:**   * The insertion of the second record should fail with an error because the UNIQUE constraint does not allow duplicate values. | I try to added an existed email and it gave me an error |
|  | Default | **DEFAULT Constraint (Nationality)**  **Description:**   * In this test, I will verify the **DEFAULT value** behavior of the **Nationality** attribute in the **Citizen** table. * I will insert a new record **without providing a value for Nationality**.   **Expected Output:**   * The record should be inserted successfully. * The **Nationality** column should automatically be set to **'jordanian'**. | I will add record without nationality :    I put Jordanian by default: |
|  | Not null | **Description:**   * In this test, I will attempt to insert a **NULL** value into the **EmployeeLevel** attribute in the **Employee** table. * The EmployeeLevel attribute is defined with a **NOT NULL** constraint.   **Expected Output:**   * The insertion should fail with an error because **NULL values are not allowed** for attributes that have a NOT NULL constraint. |  |
|  | Check | **Description:**   * In this test, I will verify the **CHECK constraint** on the **AmountPaid** attribute in the **Payment** table. * The AmountPaid attribute is constrained to accept values **greater than 100**, so I will attempt to insert the value **12**.   **Expected Output:**   * The insertion should fail with an error because the value **12 violates the CHECK constraint**. |  |
|  | Data types | **Description:**   * In this test, I will verify the **data type constraint** on the **AmountPaid** attribute in the **Payment** table. * The AmountPaid attribute is defined as **DECIMAL(10,2)**, so it must contain a numeric value.   **Expected Output:**   * The insertion should fail with an error because the provided value does **not match the required numeric data type**. |  |

## Output Validation

This section n provides evidence that the database outputs accurate and reliable results based on the implemented queries. Output validation ensures that the system retrieves correct data, follows business rules, and behaves as expected when users interact with the database.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Query Description** | **Screenshot (query + result)** | **Result validation** |
|  | This query displays the **citizen (donor)** information together with their **payment (donation)** information in one result, so we don’t need to open the tables separately. |  | This is the tables before join:    And I used all this columns for the join:    And when we compound them, this is the result:    Each **CitizenID** in the result exists in the **Citizen** table, and each **PaymentID** and **AmountPaid** matches the corresponding records in the **Payment** table through the **ServiceRequest** relationship. This confirms that the JOIN correctly combines citizen (donor) information with their payment (donation) details in one result. |
|  | This query shows the **citizens who paid more than in total700** (sum of all their payments). It helps identify the **top payers** in the system.. |  | Only citizens whose **total completed payments exceed 1000** should appear, while citizens with totals **≤ 1000** should not appear. |
|  | * This query shows how many employees in the **Employee** table have the same **job role**. * Every organization must keep a record of employee roles because it is important to know how many employees work in each position to support correct staffing and management. |  | Checking the employee table to validate the result: |
|  | This query updates the **phone number** of the employee with **EmployeeID = 102** in the **EmployeePhone** table.  This query is important because employees may change their phone numbers, and the system must be updated to keep contact information accurate. |  | This table before update    And this after update: |

## Security Validation

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Username** | **Description of privilege/no privilege** | **Screenshot (query + result)** |
|  | Citizen | The citizen user is allowed to execute specific stored procedures only. These procedures allow the citizen to view their own request information, such as the number of renewal requests and request details. The citizen cannot directly access tables, and cannot perform INSERT, UPDATE, or DELETE operations. | . |
|  | Admin | This privilege gives the admin ability to add , delete, insert ,update , select new row I will add new row in Citizen table and see if will allow me to do it . and I gave this privilege to admin because he must have an ability to manage the tables and new values in it |  |
|  | Employee | Employee can select the Citizen table from and see the values |  |
|  | Service Editor | Service Editor has privilege to update in the service table because his responsibility to manage the service so I will test if I can update on event table using Service Editor connection |  |
|  | Employee | Employee don’t have privilege to delete in the database |  |
|  | Employee | Employee don’t have privilege to add anew citizen in the database |  |

### 

## GUI Validation

This section should demonstrate that the graphical user interface (GUI) of the system has been tested to ensure it functions correctly and aligns with user and system requirements.

|  |  |  |
| --- | --- | --- |
| **Number** | **Description** | **screenshot** |
|  | The administrator uses this page to connect the system to the MySQL database by entering the required server and database credentials. |  |
|  | To test the validation the output values from view and test if we can see the information and I will compare between GUI and Query from the database to see if the output from them is the same. |  |
|  | Now I will test if adding new row in GUI will effect on the database    Now we add a new row and fill the info  After saving the new row, we can see that it has been added successfully to the Citizen table | Adding a new row and save it:      Checking from the database: |
|  | I will edit a row in the table through the graphical user interface (GUI), I will modify the email value  After making the change, we observed that the updated value was successfully reflected in the GUI. | Changing value:    Edited row:    Database: |
|  | I'll delete a row from the donor table via the GUI to check if it will be deleted in the in the database | Deleting a row from GUI:    Row deleted from database: |

## Assess whether meaningful data has been extracted

The database system is designed to extract data that is both **accurate and actionable** for the SANAD Personal ID Renewal service. Key queries executed include:

1. **Pending requests per employee**

SELECT EmployeeID, COUNT(\*) AS PendingRequests

FROM ServiceRequest

WHERE RequestStatus = 'Pending'

GROUP BY EmployeeID;

Provides a citizen with all past and active requests &Enables citizens and employees to track request status and history, supporting transparency

1. **Citizen request history**

SELECT CitizenID, RequestID, RequestStatus, SubmissionDate

FROM ServiceRequest

WHERE CitizenID = '123456789';

1. Provides a citizen with all past and active requests & Enables citizens and employees to track request status and history, supporting transparency.
2. **Payment validation**

SELECT SR.RequestID, P.AmountPaid, P.PaymentStatus

FROM ServiceRequest SR

JOIN Payment P ON SR.RequestID = P.RequestID

WHERE P.PaymentStatus = 'Completed';

Confirms which service requests are fully paid & Ensures that service completion aligns with actual payment records.

These examples demonstrate that the system can generate relevant and meaningful information for decision-making, tracking, auditing, and citizen support. The data extracted aligns directly with the system objectives, ensuring operational efficiency and accountability.

## Assess the effectiveness of testing

Testing was carried out across multiple layers:

1. **Data validation:**
   * Primary key and foreign key constraints ensured **data integrity**.
   * Tests confirmed that **multi-valued attributes** (like phone numbers) and composite attributes were normalized correctly.
2. **Output validation:**
   * Query outputs were checked against expected results using **test data** including multiple citizens, service requests, and payment records.
   * Examples included testing request status updates, payment verification, and citizen request history.
3. **Security testing:**
   * Users with limited privileges (e.g., citizen, service auditor) were unable to perform unauthorized actions such as deleting records.
   * Employees could update request status but not modify unrelated citizen data.
4. **GUI validation:**
   * Screens for request submission, payment entry, and status tracking were tested for **usability and accuracy**.

**Choice of test data:**

* Data included **multiple citizens with varying numbers of service requests**, employees handling different workloads, and payments with different statuses.
* This diversity ensured tests captured **realistic scenarios** and edge cases.

Overall, the testing process verified that the system operates reliably, produces correct outputs, and enforces both functional and security requirements.

Evaluation of database solution

## Effectiveness of the database solution based on user and system requirement

This The implemented database effectively meets both **user and system requirements**:

1. **Citizen requirements:** Can register, submit requests, track status, and make payments.
2. **Employee requirements:** Can view, update, and manage requests with accountability.
3. **Administrator requirements:** Full access for management and reporting.
4. **System requirements:** Maintains data consistency, ensures referential integrity, prevents duplicate requests, and enforces role-based access control.

The relationships between **Citizen, ServiceRequest, Employee, and Payment** tables support efficient tracking and reporting, ensuring the system meets operational needs.

## Suggested improvements

1. **Enhanced reporting features**

* Add analytics dashboards showing trends in request processing times or payment delays.

1. **Automated notifications**

* Email or SMS alerts to citizens when request status changes or payments are confirmed.

1. **Audit trail extension**

* Record all changes made by employees, with timestamps, for enhanced accountability.

1. **Performance optimization**

* Add additional indexes on frequently queried columns like RequestStatus and EmployeeID to improve query speed.

1. **Scalability improvements**

* Implement partitioning or archiving old service requests to maintain performance as data grows.

## Evaluation based on improvements needed

Implementing these improvements would **strengthen the system**:

* **Reporting and notifications:** Improve transparency and citizen engagement.
* **Extended audit trails:** Enhance security and accountability for government compliance.
* **Performance optimization:** Ensure consistent response times under higher loads.
* **Scalability:** Prepare the system for future expansions or integration with other SANAD services.

These enhancements would make the system more user-friendly, efficient, secure, and maintainable while aligning with organizational goals.

# References

1. Banister, Rick. “Physical Database Design | Sesame Software.” *Sesame Software*, 2019, sesamesoftware.com/white-papers/physical-database-design/.
2. Collins, Emily. “A Comprehensive Handbook for Regular Database Maintenance -.” *The 9th Door*, 12 Oct. 2023, the9thdoor.com/guide-to-routine-database-maintenance/?utm\_source=chatgpt.com. Accessed 21 Jan. 2025.
3. Dimitrov, Miroslav. “Tips and Tricks for SQL Server Database Maintenance Optimization.” *SQL Shack - Articles about Database Auditing, Server Performance, Data Recovery, and More*, 11 Jan. 2016, www.sqlshack.com/tips-and-tricks-for-sql-server-database-maintenance-optimization/?utm\_source=chatgpt.com. Accessed 21 Jan. 2025.
4. Nayak, Shruti. “Database Backup and Recovery Techniques - Shruti Nayak - Medium.” *Medium*, Medium, 2 May 2023, [medium.com/@shruti.nayak20/database-backup-and-recovery-techniques-c541faf3444a](mailto:medium.com/@shruti.nayak20/database-backup-and-recovery-techniques-c541faf3444a).
5. NetApp. “What Is Backup and Recovery? - Why It’s Important | NetApp.” *Www.netapp.com*, 2023, [www.netapp.com/cyber-resilience/data-protection/data-backup-recovery/what-is-backup-recovery/](http://www.netapp.com/cyber-resilience/data-protection/data-backup-recovery/what-is-backup-recovery/).
6. Seis, David. “Essential Database Maintenance Tasks Every DBA Should Know.” *SQL Server Consulting - Straight Path Solutions*, 23 Aug. 2023, straightpathsql.com/archives/2023/08/essential-database-maintenance-tasks-every-dba-should-know/?utm\_source=chatgpt.com. Accessed 21 Jan. 2025.
7. Veritas. “Data Backup & Recovery: The Essential Guide for Businesses | Veritas.” *Www.veritas.com*, 2023, [www.veritas.com/information-center/data-backup-and-recovery](http://www.veritas.com/information-center/data-backup-and-recovery).
8. “What Is Backup and Recovery?” *Cohesity*, [www.cohesity.com/glossary/backup-and-recovery/](http://www.cohesity.com/glossary/backup-and-recovery/).