**CHAPTER 1**

# INTRODUCTION

## Purpose

The purpose of this project is to design and implement a database management system for **Shree Kshethra Dharmasthala Rural Development** (S.K.D*)* initiatives. S.K.D is a real-world trust that manages various rural development programs, financial support services, and community upliftment schemes. Managing such a vast set of operations manually can result in data inconsistency, slow updates, and operational inefficiency.

This system is developed to help the trust efficiently manage its databases by organizing records of members, loan details, schemes, repayments, staff, and regional offices. It ensures a centralized platform where information can be stored, accessed, and updated in a reliable, secure, and consistent manner.

Using database triggers, relationships, and stored procedures, the system helps in automating repetitive tasks, tracking loan repayments, generating overdue notices, and maintaining data integrity. This structured digital solution contributes to better transparency, reduced manual errors, and improved service delivery by the trust.

## Scope

The scope of this project is to build a secure and scalable database system that supports various modules of S.K.D’s rural development efforts. The system targets operations such as member registration, loan sanctioning, tracking repayment schedules, managing scheme participation, and staff administration.

The database backend is developed using Microsoft SQL Server, and it is connected to a lightweight frontend built using the Flask Python framework. The system supports CRUD operations, validation through triggers, and performance optimization through indexing and normalization.

This application is intended for internal use by the trust’s administrative staff and can later be extended with features like web-based dashboards, SMS notifications, or analytics for evaluating developmental impact across regions.

.

## System Overview

The S.K.D Rural Development Database Management System addresses the need for structured and efficient data handling within the organization. It consists of tables and relationships that map real-world entities like members, loans, offices, and staff.

With Flask as the interface and MS SQL Server as the backend, the system enables seamless communication and operation handling. Key components include stored procedures for loan approval workflows, triggers for overdue tracking, and reports generation.

This system enforces data integrity, avoids duplication, and simplifies reporting and auditing for financial compliance. It is modular in design, supporting scalability for future additions like mobile access or multilingual support.

**CHAPTER 2**

# REQUIREMENT SPRECIFICATION

## Hardware Specification

* + 1. **Processor**: Intel Core i3 or higher
    2. **RAM**: Minimum 4 GB
    3. **Storage**: Minimum 10 GB free disk space
    4. **Display**: Standard desktop or laptop screen
    5. **Keyboard & Mouse**: For input operations
    6. **Network**: Required for Flask-based frontend to interact with MS SQL Server

## Software Specification

* + 1. **Operating System**: Windows 10 or later
    2. **Backend**: Microsoft SQL Server
    3. **Frontend**: Python with Flask Framework
    4. **IDE**: Visual Studio Code or PyCharm
    5. **Database Connector**: pyodbc for Python to SQL Server connectivity
    6. **Language Used**: SQL, Python (Flask)
    7. **Connectivity**: ODBC Driver with pyodbc in Flask to connect frontend to MS SQL Server
    8. **Other Tools**: SQL Server Management Studio (SSMS) for database management

**CHAPTER 3**

# SYSTEM DESIGN

## ER Diagram

As shown in Figure 3.1, he Below is the ER diagram of the system representing the key entities and their relationships involved in managing members, groups, savings, and administrative access.

A diagram of a company

AI-generated content may be incorrect.

**Fig.3.1 ER Diagram**

## Mapping from ER Diagram to Schema Diagram

The transition from the ER diagram to the relational schema involves converting entities and their relationships into normalized database tables with defined keys and constraints. Below is the detailed mapping:

### Director

* + - * Mapped as a table with DID as the Primary Key
      * Attributes: DName
      * Represents top-level administrators overseeing rural development programs

### Supervisor

* + - * Mapped as a table with Supervisor\_ID as the Primary Key
      * Attributes: Supervisor\_Name, DID as Foreign Key referencing Director(DID)
      * Each supervisor works under a specific director

### Head

* + - * Mapped as a table with HID as the Primary Key
      * Attributes: HName, Supervisor\_ID as Foreign Key referencing Supervisor(Supervisor\_ID)
      * Represents the head users who manage login and group activities

### Group

* + - * Mapped as a table with Group\_ID as the Primary Key
      * Attributes: Group\_Name, Group\_Head as Foreign Key referencing Head(HID)
      * Organizes members into working groups under a specific head

### Member

* + - * Mapped as a table with Member\_ID as the Primary Key
      * Attributes: MName, PhoneNo, Address, Age, Gender, Group\_ID as Foreign Key referencing Groups(Group\_ID)
      * Represents individuals who are part of a group and benefit from SKD services

### Savings

* + - * Mapped as a table with Member\_ID as both Primary Key and Foreign Key referencing Member(Member\_ID)
      * Attributes: Savings\_Amount, Savings\_Per\_Week
      * Stores cumulative savings and per-week contribution for each member

### Loan

* + - * Mapped as a table with Member\_ID as both Primary Key and Foreign Key referencing Member(Member\_ID)
      * Attributes: Loan\_Amount, EMI, Amount\_Paid, Amount\_Remaining
      * Captures loan status and repayment progress per member

### Suraksha

* + - * Mapped as a table with Member\_ID as Primary Key and Foreign Key referencing Member(Member\_ID)
      * Attributes: Suraksha\_No, Group\_ID (FK), Amount\_Paid, No\_Of\_Dependents, Coverage\_Limit, StartDate, EndDate
      * Represents insurance-like coverage details for each member

### Dependent

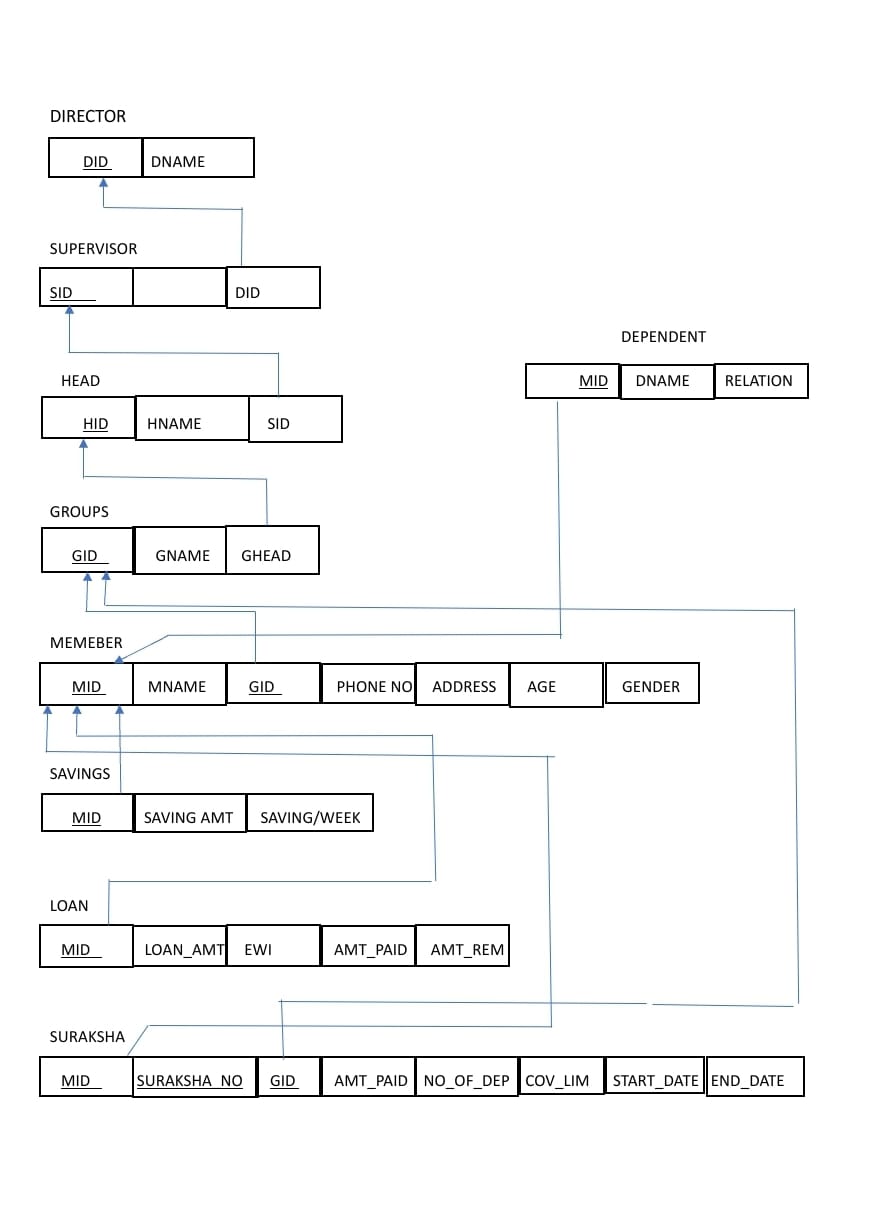
* + - * Mapped as a table with a Composite Primary Key: (Member\_ID, DName)
      * Attributes: Relation
      * Member\_ID is a Foreign Key referencing Member(Member\_ID)
      * Stores dependent details (like spouse, children, parents) of each member

## Assumptions

* + 1. The **Director** supervises one or more **Supervisors**. Every **Supervisor** is associated with exactly one **Director**
    2. Each **Head** operates under a single **Supervisor** and manages one or more **Groups**. Every **Group** is assigned to one **Head** and contains multiple **Members**.
    3. **Member** belongs to only one **Group**, but can have associated records in **Savings**, **Loan**, **Suraksha**, and **Dependent** tables
    4. Each **Member** can have only one active record in **Savings**, **Loan**, and **Suraksha** tables.
    5. A **Member** may have multiple **Dependents**, but each dependent name (DName) must be unique for that member.
    6. **Suraksha** policies are group-based, so each policy also links to the **Group\_ID**.
    7. All dates (like policy start/end) are assumed to be updated manually or via admin input.
    8. Only valid, authenticated **Head users** (HID login) can perform operations on members and financial records.
    9. No duplicate Member\_ID, Group\_ID, or Loan/Policy identifiers are allowed in the system.
    10. Phone numbers and other identifiers are assumed to be correctly formatted and validated through the frontend.

## Schema Diagram

As shown in Figure 3.4, the schema diagram provides a detailed structure of the database tables, including their attributes and foreign key connections. It supports the implementation of the system by outlining how related data is organized and maintained.



**Fig.3.4 Schema Diagram**

**CHAPTER 4**

## Pseudocode Used

# IMPLEMENTATION

* + 1. Procedure Name: UpdateSavings

Purpose: To update a member’s savings amount by adding the latest deposit. Pseudo code:

Procedure UpdateSavings(Member\_ID, DepositAmount)

Begin

If Member\_ID exists in Savings Then

Update Savings

Set Savings\_Amount = Savings\_Amount + DepositAmount

Else

Insert new record into Savings with initial DepositAmount

End

* + 1. Procedure Name: UpdateLoanPayment

Purpose: To update a loan record when a member pays an EMI.

Pseudo code:

Procedure UpdateLoanPayment(Member\_ID, EMI\_Amount)

Begin

Fetch current Amount\_Paid and Amount\_Remaining

Set Amount\_Paid = Amount\_Paid + EMI\_Amount

Set Amount\_Remaining = Amount\_Remaining - EMI\_Amount

Update Loan Table

End

* + 1. Procedure Name: AddDependent

Purpose: To insert a dependent for a member.

Pseudo code:

Procedure AddDependent(Member\_ID, DName, Relation)

Begin

If Combination(Member\_ID, DName) not exists Then

Insert into Dependent Table

Else

Raise Error: 'Dependent already exists for this member'

End

* + 1. Procedure Name: InsertSurakshaRecord

Purpose: Inserts a new Suraksha policy record for a member and sets the policy's start date to the current date

Pseudo Code:

CREATE PROCEDURE InsertSurakshaRecord

@Member\_ID VARCHAR(10),

@Suraksha\_No VARCHAR(10),

@Group\_id VARCHAR(10),

@Amount\_Paid REAL,

@No\_Of\_Dependants INT

AS

BEGIN

INSERT INTO Suraksha (Member\_ID, Suraksha\_No, Group\_id, Amount\_Paid, No\_Of\_Dependants, start\_date)

VALUES (@Member\_ID, @Suraksha\_No, @Group\_id, @Amount\_Paid, @No\_Of\_Dependants, GETDATE());

END;

* + 1. Trigger Name: trg\_SetEndDate

Purpose: Automatically sets the end\_date of the Suraksha policy to one year after the start\_date when a new record is inserted.

Pseudo Code:

CREATE TRIGGER trg\_SetEndDate

ON Suraksha

AFTER INSERT

AS

BEGIN

UPDATE S

SET

start\_date = ISNULL(I.start\_date, GETDATE()),

end\_date = DATEADD(YEAR, 1, ISNULL(I.start\_date, GETDATE()))

FROM Suraksha S

INNER JOIN inserted I

ON S.Member\_ID = I.Member\_ID AND S.Suraksha\_No = I.Suraksha\_No;

END;

## Tables Used

**1. Director**

* **Purpose**: Top-level administrators
* **Key Columns**: DID (PK), DName

**2. Supervisor**

* **Purpose**: Mid-level managers under a Director
* **Key Columns**: Supervisor\_ID (PK), Supervisor\_Name, DID (FK)

**3. Head**

* **Purpose**: System users responsible for managing Groups
* **Key Columns**: HID (PK), HName, Supervisor\_ID (FK)

**4. Groups**

* **Purpose**: Functional units within the rural framework
* **Key Columns**: Group\_ID (PK), Group\_Name, Group\_Head (FK to Head)

**5. Member**

* **Purpose**: Individual members enrolled in SKD programs
* **Key Columns**: Member\_ID (PK), MName, PhoneNo, Address, Age, Gender, Group\_ID (FK)

**6. Savings**

* **Purpose**: Holds member savings info
* **Key Columns**: Member\_ID (PK, FK), Savings\_Amount, Savings\_Per\_Week

**7. Loan**

* **Purpose**: Tracks loan status of a member
* **Key Columns**: Member\_ID (PK, FK), Loan\_Amount, EMI, Amount\_Paid, Amount\_Remaining

**8. Suraksha**

* **Purpose**: Insurance-like coverage per member
* **Key Columns**: Member\_ID (PK, FK), Suraksha\_No, Group\_ID (FK), Amount\_Paid, No\_Of\_Dependents, Coverage\_Limit, StartDate, EndDate

**9. Dependent**

* **Purpose**: Dependent family members of a member
* **Key Columns**: Composite PK (Member\_ID, DName), Relation

**CHAPTER 5**

# RESULTS AND DISCUSSION

## Results

* + 1. The system successfully handled **member registration**, **group association**, and **hierarchical user access** (Director → Supervisor → Head).
    2. Suraksha policy data was inserted using a stored procedure that automatically populated the start date.
    3. Savings and loan records were managed accurately through update procedures, keeping totals consistent and avoiding negative balances.
    4. A trigger (trg\_SetEndDate) worked as expected, setting the end date to exactly **one year** from the start date.
    5. Group-wise and member-wise organization allowed for **efficient data retrieval**, ensuring clarity in reporting and traceability.
    6. **Authentication** via the Head login ensured that only authorized personnel could access or modify the database.
    7. The system maintained **relational integrity** through the use of foreign keys and constraints across the database schema.

1. Real-time communication between the Flask frontend and SQL Server backend was established using pyodbc, making the application interactive and dynamic.

## Discussion

1. The InsertSurakshaRecord stored procedure simplified inserting new policy records and avoided missing start dates.
2. The trg\_SetEndDate trigger ensured data accuracy by automatically calculating and populating the policy end date.
3. Proper use of foreign keys maintained consistency across the system, especially between **Member**, **Group**, and **Suraksha** tables.
4. Error handling was implemented to prevent duplicate dependents for a single member and disallow negative loan balances.
5. One initial challenge was **handling composite keys** in the Dependent table, which was resolved using a compound primary key (Member\_ID, DName).
6. The hierarchical structure added complexity but enhanced **data segregation**, accountability, and traceability across different administrative levels.
7. Frontend validation using Flask combined with SQL-side constraints helped maintain **data quality** and **user control**.

**CHAPTER 6**

# CONCLUSION AND FUTURE WORK

## Conclusion

The **SKD Rural Development Information System** was successfully designed and implemented using Microsoft SQL Server for the backend and Python Flask for the frontend interface. The system focused on organizing and managing records related to **members, groups, savings, loans, dependents**, and **Suraksha insurance policies** under a real-world rural development trust.

Stored procedures and triggers were used to automate tasks such as savings updates and policy date tracking. The hierarchical structure involving Directors, Supervisors, Heads, and Groups ensured a well-structured data access model. The application maintained data integrity, avoided redundancy, and offered secure and efficient data operations through session-based authentication.

The implementation met all functional requirements such as:

* Registering members under specific groups
* Tracking savings and loans per member
* Managing Suraksha policy records with auto-set start and end dates
* Inserting dependents with enforced uniqueness
* Real-time interactions via a Flask web interface connected using ODBC

## Future Work

1. **Graphical Dashboard**: Integrate charts and visual reports for savings, loans, and policy status.
2. **Role-Based Authentication**: Expand user roles (Director, Supervisor, Head) with restricted access controls.
3. **Mobile App Support**: Extend the frontend to support mobile browsers or Android apps for field usage.
4. **SMS/Email Alerts**: Notify members about loan dues, policy expirations, and savings updates via communication tools.
5. **Report Generator**: Add downloadable reports in PDF/Excel formats for administrative use.
6. **Search & Filter**: Implement advanced search and filtering features for quick data access.

# REFERENCES

1. Microsoft SQL Server Documentation – <https://learn.microsoft.com/sql>
2. Flask Web Framework Documentation – https://flask.palletsprojects.com
3. pyodbc Python Library – <https://pypi.org/project/pyodbc>
4. SQL Triggers and Stored Procedures Guide – <https://www.sqlservertutorial.net>
5. OpenAI ChatGPT – <https://chat.openai.com> — used for conceptual guidance and pseudocode writing
6. YouTube Tutorials – SQL Server & Flask Integration – <https://www.youtube.com>
7. SKDRDP Official Website – <https://www.skdrdpindia.org> *(for trust structure and inspiration)*