MySQL Database Setup for Transport Cost Evaluation System

Step 1: Database Schema Creation

1.1 Create Database and Tables

sql

```sql
-- Create the database
CREATE DATABASE IF NOT EXISTS Transport_database;
USE Transport_database;

-- Materials table
CREATE TABLE IF NOT EXISTS materials (
    material_id INT AUTO_INCREMENT PRIMARY KEY,
    project_name VARCHAR(100),
    material_no VARCHAR(50) UNIQUE NOT NULL,
    material_desc VARCHAR(200),
    weight_per_pcs DECIMAL(10, 4),
    usage_desc VARCHAR(200),
    daily_demand DECIMAL(10, 2),
    annual_volume INT,
    lifetime_volume DECIMAL(15, 2),
    peak_year VARCHAR(20),
    peak_year_volume INT,
    working_days INT DEFAULT 250,
    sop VARCHAR(50),
    pcs_price DECIMAL(10, 4),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_material_no (material_no)
);

-- Suppliers table
CREATE TABLE IF NOT EXISTS suppliers (
    supplier_id INT AUTO_INCREMENT PRIMARY KEY,
    vendor_id VARCHAR(20) UNIQUE NOT NULL,
    vendor_name VARCHAR(100),
    vendor_country VARCHAR(50),
    city_of_manufacture VARCHAR(100),
    vendor_zip VARCHAR(20),
    delivery_performance DECIMAL(5, 2),
    deliveries_per_month INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_vendor_id (vendor_id)
);

-- Locations table
CREATE TABLE IF NOT EXISTS locations (
    location_id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    plant VARCHAR(100) UNIQUE NOT NULL,
    country VARCHAR(50),
    distance DECIMAL(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Operations table
CREATE TABLE IF NOT EXISTS operations (
    operation_id INT AUTO_INCREMENT PRIMARY KEY,
    incoterm_code VARCHAR(10),
    incoterm_place VARCHAR(100),
    part_class VARCHAR(20),
    calloff_type VARCHAR(50),
    directive VARCHAR(3),
    lead_time INT,
    subsupplier_used VARCHAR(3),
    subsupplier_box_days INT,
    packaging_tool_owner VARCHAR(50),
    responsible VARCHAR(50),
    currency VARCHAR(3),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Packaging configurations table
CREATE TABLE IF NOT EXISTS packaging_configs (
    packaging_id INT AUTO_INCREMENT PRIMARY KEY,
    pack_maint DECIMAL(10, 4),
    empties_scrap DECIMAL(10, 4),
    box_type VARCHAR(50),
    fill_qty_box INT,
    pallet_type VARCHAR(50),
    fill_qty_lu_oversea INT,
    add_pack_price DECIMAL(10, 2),
    sp_needed VARCHAR(3),
    sp_type VARCHAR(50),
    fill_qty_tray INT,
    tooling_cost DECIMAL(10, 2),
    add_sp_pack VARCHAR(3),
    trays_per_sp_pal INT,
    sp_pallets_per_lu INT,
    loop_data JSON,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Transport lanes table (from Transport cost.xlsx)
CREATE TABLE IF NOT EXISTS transport_lanes (
    lane_id INT AUTO_INCREMENT PRIMARY KEY,
    origin_country VARCHAR(5),
    origin_zip VARCHAR(10),
    origin_city VARCHAR(100),
    dest_country VARCHAR(5),
    dest_zip VARCHAR(10),
    dest_city VARCHAR(100),
    lane_code VARCHAR(20) UNIQUE,
    shipments_per_year INT,
    weight_per_year DECIMAL(15, 2),
    avg_shipment_size DECIMAL(10, 2),
    lead_time_groupage VARCHAR(10),
    lead_time_ltl VARCHAR(10),
    lead_time_ftl VARCHAR(10),
    full_truck_price DECIMAL(10, 2),
    fuel_surcharge DECIMAL(5, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_lane_code (lane_code),
    INDEX idx_origin (origin_country, origin_zip),
    INDEX idx_dest (dest_country, dest_zip)
);

-- Transport prices by weight table
CREATE TABLE IF NOT EXISTS transport_prices (
    price_id INT AUTO_INCREMENT PRIMARY KEY,
    lane_id INT,
    weight_limit DECIMAL(10, 2),
    price DECIMAL(10, 2),
    FOREIGN KEY (lane_id) REFERENCES transport_lanes(lane_id) ON DELETE CASCADE,
    INDEX idx_lane_weight (lane_id, weight_limit)
);

-- Weight clusters lookup table
CREATE TABLE IF NOT EXISTS weight_clusters (
    cluster_id INT AUTO_INCREMENT PRIMARY KEY,
    weight_limit DECIMAL(10, 2) UNIQUE,
    cluster_name VARCHAR(50)
);
```

```sql
-- Insert standard weight clusters
INSERT INTO weight_clusters (weight_limit, cluster_name) VALUES
(50, '≤50kg'),
(75, '≤75kg'),
(100, '≤100kg'),
(150, '≤150kg'),
(200, '≤200kg'),
(300, '≤300kg'),
(400, '≤400kg'),
(500, '≤500kg'),
(600, '≤600kg'),
(700, '≤700kg'),
(800, '≤800kg'),
(900, '≤900kg'),
(1000, '≤1000kg'),
(1500, '≤1500kg'),
(2000, '≤2000kg'),
(2500, '≤2500kg'),
(3000, '≤3000kg'),
(5000, '≤5000kg'),
(7500, '≤7500kg'),
(10000, '≤10000kg'),
(15000, '≤15000kg'),
(20000, '≤20000kg'),
(22500, '≤22500kg');

-- Warehouse configurations table
CREATE TABLE IF NOT EXISTS warehouse_configs (
    warehouse_id INT AUTO_INCREMENT PRIMARY KEY,
    cost_per_loc DECIMAL(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- CO2 configurations table
CREATE TABLE IF NOT EXISTS co2_configs (
    co2_id INT AUTO_INCREMENT PRIMARY KEY,
    cost_per_ton DECIMAL(10, 2),
    co2_conversion_factor DECIMAL(5, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

```sql
-- Customs configurations table
CREATE TABLE IF NOT EXISTS customs_configs (
    customs_id INT AUTO_INCREMENT PRIMARY KEY,
    pref_usage VARCHAR(3),
    duty_rate DECIMAL(5, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

-- Material-Supplier associations table
CREATE TABLE IF NOT EXISTS material_supplier_pairs (
    pair_id INT AUTO_INCREMENT PRIMARY KEY,
    material_no VARCHAR(50),
    vendor_id VARCHAR(20),
    active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (material_no) REFERENCES materials(material_no) ON DELETE CASCADE,
    FOREIGN KEY (vendor_id) REFERENCES suppliers(vendor_id) ON DELETE CASCADE,
    UNIQUE KEY unique_pair (material_no, vendor_id),
    INDEX idx_active_pairs (active)
);

-- Calculation results storage table
CREATE TABLE IF NOT EXISTS calculation_results (
    result_id INT AUTO_INCREMENT PRIMARY KEY,
    material_no VARCHAR(50),
    vendor_id VARCHAR(20),
    packaging_cost_per_piece DECIMAL(10, 6),
    transport_cost_per_piece DECIMAL(10, 6),
    warehouse_cost_per_piece DECIMAL(10, 6),
    customs_cost_per_piece DECIMAL(10, 6),
    co2_cost_per_piece DECIMAL(10, 6),
    repacking_cost_per_piece DECIMAL(10, 6),
    additional_cost_per_piece DECIMAL(10, 6),
    total_cost_per_piece DECIMAL(10, 6),
    total_annual_cost DECIMAL(15, 2),
    calculation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    calculation_params JSON,
    FOREIGN KEY (material_no) REFERENCES materials(material_no) ON DELETE CASCADE,
    FOREIGN KEY (vendor_id) REFERENCES suppliers(vendor_id) ON DELETE CASCADE,
    INDEX idx_calc_date (calculation_date),
    INDEX idx_material_vendor (material_no, vendor_id)
);
```

```sql
-- User scenarios table for saving/loading calculation scenarios
CREATE TABLE IF NOT EXISTS calculation_scenarios (
    scenario_id INT AUTO_INCREMENT PRIMARY KEY,
    scenario_name VARCHAR(100),
    scenario_description TEXT,
    configuration_data JSON,
    created_by VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_scenario_name (scenario_name),
    INDEX idx_created_by (created_by)
);
```

1.2 Create Stored Procedures for Complex Calculations

sql

```
DELIMITER //

-- Stored procedure to calculate transport cost based on weight and route
CREATE PROCEDURE CalculateTransportCost(
    IN p_origin_country VARCHAR(5),
    IN p_origin_zip VARCHAR(10),
    IN p_dest_country VARCHAR(5),
    IN p_dest_zip VARCHAR(10),
    IN p_weight DECIMAL(10, 2),
    IN p_pallets INT,
    IN p_loading_meters DECIMAL(10, 2),
    OUT p_cost DECIMAL(10, 2),
    OUT p_cost_type VARCHAR(20)
)
BEGIN
    DECLARE v_lane_id INT;
    DECLARE v_weight_based_cost DECIMAL(10, 2);
    DECLARE v_space_based_cost DECIMAL(10, 2);
    DECLARE v_full_truck_price DECIMAL(10, 2);
    DECLARE v_fuel_surcharge DECIMAL(5, 2);
    DECLARE v_is_international BOOLEAN;

    -- Find the lane
    SELECT lane_id, full_truck_price, fuel_surcharge
    INTO v_lane_id, v_full_truck_price, v_fuel_surcharge
    FROM transport_lanes
    WHERE origin_country = p_origin_country
      AND LEFT(origin_zip, 2) = LEFT(p_origin_zip, 2)
      AND dest_country = p_dest_country
      AND LEFT(dest_zip, 2) = LEFT(p_dest_zip, 2)
    LIMIT 1;

    -- Check if international
    SET v_is_international = (p_origin_country != p_dest_country);

    -- Get weight-based cost
    SELECT price INTO v_weight_based_cost
    FROM transport_prices
    WHERE lane_id = v_lane_id
      AND weight_limit >= p_weight
    ORDER BY weight_limit ASC
    LIMIT 1;
```

```
    -- Calculate space-based cost
    IF v_is_international THEN
        SET v_space_based_cost = p_loading_meters * 1500;
    ELSE
        SET v_space_based_cost = p_loading_meters * 800;
    END IF;

    -- Determine which cost to use
    IF p_pallets > 34 AND v_full_truck_price IS NOT NULL THEN
        -- Full truck load calculation
        SET p_cost = v_full_truck_price;
        SET p_cost_type = 'full_truck';
    ELSEIF v_space_based_cost > v_weight_based_cost THEN
        SET p_cost = v_space_based_cost;
        SET p_cost_type = 'space_based';
    ELSE
        SET p_cost = v_weight_based_cost;
        SET p_cost_type = 'weight_based';
    END IF;

    -- Apply fuel surcharge if exists
    IF v_fuel_surcharge IS NOT NULL THEN
        SET p_cost = p_cost * (1 + v_fuel_surcharge / 100);
    END IF;
END//

-- Stored procedure for complete logistics cost calculation
CREATE PROCEDURE CalculateTotalLogisticsCost(
    IN p_material_no VARCHAR(50),
    IN p_vendor_id VARCHAR(20),
    OUT p_total_cost DECIMAL(10, 6),
    OUT p_annual_cost DECIMAL(15, 2)
)
BEGIN
    DECLARE v_annual_volume INT;
    DECLARE v_packaging_cost DECIMAL(10, 6) DEFAULT 0;
    DECLARE v_transport_cost DECIMAL(10, 6) DEFAULT 0;
    DECLARE v_warehouse_cost DECIMAL(10, 6) DEFAULT 0;
    DECLARE v_customs_cost DECIMAL(10, 6) DEFAULT 0;
    DECLARE v_co2_cost DECIMAL(10, 6) DEFAULT 0;

    -- Get annual volume
    SELECT annual_volume INTO v_annual_volume
    FROM materials
```

```
    WHERE material_no = p_material_no;

    -- Calculate individual cost components (simplified - expand as needed)
    -- These would call other procedures or functions for detailed calculations

    -- Sum up total cost per piece
    SET p_total_cost = v_packaging_cost + v_transport_cost +
                       v_warehouse_cost + v_customs_cost + v_co2_cost;

    -- Calculate annual cost
    SET p_annual_cost = p_total_cost * v_annual_volume;

    -- Store the result
    INSERT INTO calculation_results (
        material_no, vendor_id,
        packaging_cost_per_piece, transport_cost_per_piece,
        warehouse_cost_per_piece, customs_cost_per_piece,
        co2_cost_per_piece, total_cost_per_piece, total_annual_cost
    ) VALUES (
        p_material_no, p_vendor_id,
        v_packaging_cost, v_transport_cost,
        v_warehouse_cost, v_customs_cost,
        v_co2_cost, p_total_cost, p_annual_cost
    );
END//

DELIMITER ;
```

Step 2: Data Import Scripts

2.1 Python Script to Import Excel Data to MySQL

python

```python
# import_excel_to_mysql.py
import pandas as pd
import mysql.connector
from mysql.connector import Error
import json
import os
from pathlib import Path

class TransportDatabaseImporter:
    def __init__(self, host='localhost', database='Transport_database',
                 user='root', password=''):
        """Initialize database connection"""
        self.connection = None
        self.cursor = None
        try:
            self.connection = mysql.connector.connect(
                host=host,
                database=database,
                user=user,
                password=password
            )
            self.cursor = self.connection.cursor()
            print(f"Successfully connected to MySQL database: {database}")
        except Error as e:
            print(f"Error connecting to MySQL: {e}")

    def import_transport_costs(self, excel_file_path):
        """Import Transport cost.xlsx data"""
        print(f"Importing transport costs from {excel_file_path}")

        # Read Excel file
        df = pd.read_excel(excel_file_path, sheet_name=0)

        # Find header row with weight clusters
        header_row = None
        for idx, row in df.iterrows():
            if pd.notna(row.iloc[14]) and str(row.iloc[14]).startswith("≤"):
                header_row = idx
                break

        if header_row is None:
            print("Could not find header row with weight clusters")
            return
```

```
        # Extract weight clusters
        weight_clusters = []
        for col in range(14, len(df.columns)):
            value = df.iloc[header_row, col]
            if pd.notna(value) and str(value).startswith("≤"):
                weight = float(str(value).replace("≤", "").strip())
                weight_clusters.append(weight)

        # Process data rows
        for idx in range(header_row + 1, len(df)):
            row = df.iloc[idx]

            # Skip empty rows
            if pd.isna(row.iloc[0]):
                continue

            # Insert lane data
            lane_data = {
                'origin_country': str(row.iloc[1]) if pd.notna(row.iloc[1]) else "",
                'origin_zip': str(row.iloc[2]) if pd.notna(row.iloc[2]) else "",
                'origin_city': str(row.iloc[3]) if pd.notna(row.iloc[3]) else "",
                'dest_country': str(row.iloc[4]) if pd.notna(row.iloc[4]) else "",
                'dest_zip': str(row.iloc[5]) if pd.notna(row.iloc[5]) else "",
                'dest_city': str(row.iloc[6]) if pd.notna(row.iloc[6]) else "",
                'lane_code': str(row.iloc[7]) if pd.notna(row.iloc[7]) else "",
                'shipments_per_year': self._parse_number(row.iloc[8]),
                'weight_per_year': self._parse_number(row.iloc[9]),
                'avg_shipment_size': self._parse_number(row.iloc[10]),
                'lead_time_groupage': str(row.iloc[11]) if pd.notna(row.iloc[11]) else "",
                'lead_time_ltl': str(row.iloc[12]) if pd.notna(row.iloc[12]) else "",
                'lead_time_ftl': str(row.iloc[13]) if pd.notna(row.iloc[13]) else ""
            }

            # Insert lane
            insert_lane_query = """
                INSERT INTO transport_lanes
                (origin_country, origin_zip, origin_city, dest_country, dest_zip,
                 dest_city, lane_code, shipments_per_year, weight_per_year,
                 avg_shipment_size, lead_time_groupage, lead_time_ltl, lead_time_ftl)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                ON DUPLICATE KEY UPDATE
                    shipments_per_year = VALUES(shipments_per_year),
                    weight_per_year = VALUES(weight_per_year)
```

```
            """

            self.cursor.execute(insert_lane_query, (
                lane_data['origin_country'], lane_data['origin_zip'],
                lane_data['origin_city'], lane_data['dest_country'],
                lane_data['dest_zip'], lane_data['dest_city'],
                lane_data['lane_code'], lane_data['shipments_per_year'],
                lane_data['weight_per_year'], lane_data['avg_shipment_size'],
                lane_data['lead_time_groupage'], lane_data['lead_time_ltl'],
                lane_data['lead_time_ftl']
            ))

            # Get the lane_id
            lane_id = self.cursor.lastrowid
            if lane_id == 0:  # If duplicate, get existing ID
                self.cursor.execute(
                    "SELECT lane_id FROM transport_lanes WHERE lane_code = %s",
                    (lane_data['lane_code'],)
                )
                result = self.cursor.fetchone()
                if result:
                    lane_id = result[0]

            # Insert price data for each weight cluster
            for i, weight in enumerate(weight_clusters):
                price_value = row.iloc[15 + i]
                if pd.notna(price_value):
                    price = self._parse_price(price_value)
                    if price is not None:
                        insert_price_query = """
                            INSERT INTO transport_prices (lane_id, weight_limit, price)
                            VALUES (%s, %s, %s)
                            ON DUPLICATE KEY UPDATE price = VALUES(price)
                        """
                        self.cursor.execute(insert_price_query, (lane_id, weight, price))

        self.connection.commit()
        print(f"Successfully imported transport costs")

    def import_workflow_data(self, excel_file_path):
        """Import data from Mappe2 1.xlsx workflow file"""
        print(f"Importing workflow data from {excel_file_path}")

        # Read multiple sheets if needed
```

```
        xls = pd.ExcelFile(excel_file_path)

        for sheet_name in xls.sheet_names:
            df = pd.read_excel(xls, sheet_name=sheet_name)

            # Process based on sheet content
            # This would need to be customized based on actual sheet structure
            if 'material' in sheet_name.lower():
                self._import_materials(df)
            elif 'supplier' in sheet_name.lower():
                self._import_suppliers(df)
            elif 'packaging' in sheet_name.lower():
                self._import_packaging(df)
            # Add more conditions as needed

    def _import_materials(self, df):
        """Import materials data"""
        for _, row in df.iterrows():
            insert_query = """
                INSERT INTO materials
                (material_no, material_desc, weight_per_pcs, annual_volume)
                VALUES (%s, %s, %s, %s)
                ON DUPLICATE KEY UPDATE
                    material_desc = VALUES(material_desc),
                    weight_per_pcs = VALUES(weight_per_pcs)
            """
            # Adjust column names based on actual Excel structure
            self.cursor.execute(insert_query, (
                row.get('Material_No'),
                row.get('Description'),
                row.get('Weight'),
                row.get('Annual_Volume')
            ))
        self.connection.commit()

    def _import_suppliers(self, df):
        """Import suppliers data"""
        for _, row in df.iterrows():
            insert_query = """
                INSERT INTO suppliers
                (vendor_id, vendor_name, vendor_country, city_of_manufacture, vendor_zip)
                VALUES (%s, %s, %s, %s, %s)
                ON DUPLICATE KEY UPDATE
                    vendor_name = VALUES(vendor_name)
```

```
            """
            self.cursor.execute(insert_query, (
                row.get('Vendor_ID'),
                row.get('Vendor_Name'),
                row.get('Country'),
                row.get('City'),
                row.get('ZIP')
            ))
        self.connection.commit()

    def _import_packaging(self, df):
        """Import packaging configurations"""
        for _, row in df.iterrows():
            # Convert loop data to JSON
            loop_data = {
                'goods_receipt': row.get('goods_receipt', 0),
                'stock_raw_materials': row.get('stock', 0),
                'production': row.get('production', 0),
                # Add all loop stages
            }

            insert_query = """
                INSERT INTO packaging_configs
                (box_type, fill_qty_box, pallet_type, loop_data)
                VALUES (%s, %s, %s, %s)
            """
            self.cursor.execute(insert_query, (
                row.get('Box_Type'),
                row.get('Fill_Qty'),
                row.get('Pallet_Type'),
                json.dumps(loop_data)
            ))
        self.connection.commit()

    def _parse_number(self, value):
        """Parse numeric value"""
        if pd.isna(value):
            return None
        try:
            clean_value = str(value).replace(" ", "").replace(",", ".")
            return float(clean_value)
        except:
            return None
```

```
    def _parse_price(self, value):
        """Parse price value"""
        if pd.isna(value):
            return None
        try:
            clean_value = str(value).replace("€", "").replace(" ", "").replace(",", ".")
            return float(clean_value)
        except:
            return None

    def close(self):
        """Close database connection"""
        if self.connection and self.connection.is_connected():
            self.cursor.close()
            self.connection.close()
            print("MySQL connection closed")

# Main execution
if __name__ == "__main__":
    # Initialize importer
    importer = TransportDatabaseImporter(
        host='localhost',
        database='Transport_database',
        user='your_username',
        password='your_password'
    )

    # Import Transport cost data
    importer.import_transport_costs('Transport cost.xlsx')

    # Import workflow data
    importer.import_workflow_data('Mappe2 1.xlsx')

    # Close connection
    importer.close()
```

Step 3: MySQL Integration for Streamlit App

3.1 MySQL Data Manager Class

python

```python
# utils/mysql_manager.py
import mysql.connector
from mysql.connector import Error
import json
import pandas as pd
from typing import Dict, List, Optional, Any
import streamlit as st

class MySQLDataManager:
    """MySQL database manager for the logistics application"""

    def __init__(self):
        """Initialize MySQL connection using Streamlit secrets or config"""
        self.connection = None
        self.cursor = None
        self._connect()

    def _connect(self):
        """Establish database connection"""
        try:
            # Get credentials from Streamlit secrets or environment
            db_config = {
                'host': st.secrets.get("mysql", {}).get("host", "localhost"),
                'database': st.secrets.get("mysql", {}).get("database", "Transport_database"),
                'user': st.secrets.get("mysql", {}).get("user", "root"),
                'password': st.secrets.get("mysql", {}).get("password", "")
            }

            self.connection = mysql.connector.connect(**db_config)
            self.cursor = self.connection.cursor(dictionary=True)

        except Error as e:
            st.error(f"Error connecting to MySQL: {e}")

    def _ensure_connection(self):
        """Ensure database connection is active"""
        if not self.connection or not self.connection.is_connected():
            self._connect()

    # Material Management
    def add_material(self, material_data: Dict[str, Any]) -> bool:
        """Add a new material to the database"""
        self._ensure_connection()
```

```
        try:
            query = """
                INSERT INTO materials
                (project_name, material_no, material_desc, weight_per_pcs,
                 usage_desc, daily_demand, annual_volume, lifetime_volume,
                 peak_year, peak_year_volume, working_days, sop, pcs_price)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """
            values = (
                material_data.get('project_name'),
                material_data.get('material_no'),
                material_data.get('material_desc'),
                material_data.get('weight_per_pcs'),
                material_data.get('usage'),
                material_data.get('daily_demand'),
                material_data.get('annual_volume'),
                material_data.get('lifetime_volume'),
                material_data.get('peak_year'),
                material_data.get('peak_year_volume'),
                material_data.get('working_days', 250),
                material_data.get('sop'),
                material_data.get('Pcs_Price')
            )

            self.cursor.execute(query, values)
            self.connection.commit()
            return True

        except Error as e:
            st.error(f"Error adding material: {e}")
            return False

    def get_materials(self) -> List[Dict[str, Any]]:
        """Get all materials from database"""
        self._ensure_connection()
        try:
            query = "SELECT * FROM materials ORDER BY material_no"
            self.cursor.execute(query)
            return self.cursor.fetchall()
        except Error as e:
            st.error(f"Error fetching materials: {e}")
            return []

    def update_material(self, material_no: str, updated_data: Dict[str, Any]) -> bool:
```

```
    """Update existing material"""
    self._ensure_connection()
    try:
        query = """
            UPDATE materials
            SET project_name = %s, material_desc = %s, weight_per_pcs = %s,
                usage_desc = %s, daily_demand = %s, annual_volume = %s,
                lifetime_volume = %s, peak_year = %s, peak_year_volume = %s,
                working_days = %s, sop = %s, pcs_price = %s
            WHERE material_no = %s
        """
        values = (
            updated_data.get('project_name'),
            updated_data.get('material_desc'),
            updated_data.get('weight_per_pcs'),
            updated_data.get('usage'),
            updated_data.get('daily_demand'),
            updated_data.get('annual_volume'),
            updated_data.get('lifetime_volume'),
            updated_data.get('peak_year'),
            updated_data.get('peak_year_volume'),
            updated_data.get('working_days'),
            updated_data.get('sop'),
            updated_data.get('Pcs_Price'),
            material_no
        )

        self.cursor.execute(query, values)
        self.connection.commit()
        return True

    except Error as e:
        st.error(f"Error updating material: {e}")
        return False

def delete_material(self, material_no: str) -> bool:
    """Delete material from database"""
    self._ensure_connection()
    try:
        query = "DELETE FROM materials WHERE material_no = %s"
        self.cursor.execute(query, (material_no,))
        self.connection.commit()
        return True
    except Error as e:
```

```
        st.error(f"Error deleting material: {e}")
        return False

# Supplier Management
def add_supplier(self, supplier_data: Dict[str, Any]) -> bool:
    """Add a new supplier to the database"""
    self._ensure_connection()
    try:
        query = """
            INSERT INTO suppliers
            (vendor_id, vendor_name, vendor_country, city_of_manufacture,
             vendor_zip, delivery_performance,
```