Assessment Question

Employee Leave Application Module (with Cascading Dropdowns & Validation)

Your task is to design and implement a **Leave Application** feature for an organization that has multiple departments and employees. Each employee has a **fixed number of leaves for each leave type** (e.g., Medical, Casual, Earned, etc.).

The system must **not allow** an employee to apply for a specific leave type if they have **already exhausted** their leave balance for that type.

Functional Requirements

1. **Cascading Dropdowns (at least 3 levels)**
   You must implement **three cascading dropdowns** on the Leave Application form:
   1. **Department** (Dropdown 1)
      - Lists all departments in the organization (e.g., HR, IT, Finance, Sales, etc.).
   2. **Employee** (Dropdown 2 – depends on Department)
      - After selecting a Department, the Employee dropdown must be populated **only with employees from that department**.
   3. **Leave Type** (Dropdown 3 – depends on selected Employee)
      - After selecting an Employee, the Leave Type dropdown must display only those leave types for which that employee has a **non-zero leave balance**.
      - Example leave types: Medical Leave, Casual Leave, Earned Leave, etc.
2. **Leave Application Form Fields**
   Apart from the dropdowns above, the form must include:
   - **From Date** (date picker)
   - **To Date** (date picker)
   - **Reason for Leave** (text area / input)
3. **Business Rules & Validations**

Implement the following validations (client-side + server-side where applicable):

   1. **Mandatory Fields**
      - Department, Employee, Leave Type, From Date, To Date, and Reason are all **mandatory**.
      - Show appropriate error messages if any are missing.
   2. **Date Validation**
      - To Date must be **greater than or equal to** From Date.
      - The system should **calculate the total number of leave days** based on From and To dates (excluding weekends if you want to add complexity – optional but preferred).
   3. **Leave Balance Check (Critical)**
      - Each employee has a **predefined leave quota** per leave type for a given year (e.g., 10 Casual, 8 Medical, etc.).
      - When applying for leave:
        - Calculate the requested number of days.
        - Check how many leaves of that type the employee has already taken.
        - If the requested days **exceed the remaining balance**, the system must:
          - **Block submission** and show an error message like:

"Insufficient leave balance for selected leave type. Remaining: 2 days, Requested: 4 days."

   4. **Duplicate / Overlapping Leave Validation**
      - The system should **not allow overlapping leave applications** for the same employee and leave type.
      - If the selected date range overlaps with an already approved/pending leave request, show a validation message.

5. **Inactive / Ex-Employee Check (Extra validation)**
    - If an employee is marked as **inactive**, the system should not allow applying leave for them; the Employee dropdown should:
        - Either not show inactive employees, OR
        - Show an error if selected (depending on your design).

4. **Technical Expectations (You can adapt based on your stack)**
   The candidate should:
    - Design a **data model** (tables or classes) for:
        - Departments
        - Employees (with Department reference)
        - Leave Types (with yearly quota per employee)
        - Leave Applications (with status, dates, type, etc.)
    - Implement:
        - Cascading dropdown logic (e.g., via AJAX / API calls / event handlers).
        - Server-side validation logic for:
            - Leave balance
            - Overlapping dates
        - Client-side validations for:
            - Mandatory fields
            - Date relationships (From <= To)

Here's a clean, ready-to-use relational database structure for your leave application system.

---

1. Departments

```
CREATE TABLE Departments (
   DepartmentId      INT PRIMARY KEY IDENTITY(1,1),
   DepartmentName    VARCHAR(100) NOT NULL UNIQUE,
   IsActive          BIT NOT NULL DEFAULT 1
);
```

---

2. Employees

```
CREATE TABLE Employees (
   EmployeeId       INT PRIMARY KEY IDENTITY(1,1),
   EmployeeCode     VARCHAR(50) NOT NULL UNIQUE,
   FirstName        VARCHAR(100) NOT NULL,
   LastName         VARCHAR(100) NULL,
   DepartmentId     INT NOT NULL,
   DateOfJoining    DATE NULL,
   IsActive         BIT NOT NULL DEFAULT 1,

   CONSTRAINT FK_Employees_Departments
      FOREIGN KEY (DepartmentId) REFERENCES Departments(DepartmentId)
);
```

- **Cascading dropdown 1 → 2**
    - Dropdown 1: Departments
    - Dropdown 2: Employees filtered by DepartmentId and IsActive = 1.

---

3. LeaveTypes

```
CREATE TABLE LeaveTypes (
  LeaveTypeId      INT PRIMARY KEY IDENTITY(1,1),
  LeaveTypeCode    VARCHAR(20) NOT NULL UNIQUE,   -- e.g., CL, ML, EL
  LeaveTypeName    VARCHAR(100) NOT NULL,        -- Casual, Medical, etc.
  IsPaidLeave      BIT NOT NULL DEFAULT 1,
  IsActive         BIT NOT NULL DEFAULT 1
);
```

---

4. EmployeeLeaveQuota

Stores yearly quota and consumed leaves per employee and leave type.

```
CREATE TABLE EmployeeLeaveQuota (
  QuotaId          INT PRIMARY KEY IDENTITY(1,1),
  EmployeeId       INT NOT NULL,
  LeaveTypeId      INT NOT NULL,
  LeaveYear        INT NOT NULL,              -- e.g., 2025
  TotalAllocated   DECIMAL(5,2) NOT NULL,      -- e.g., 10.00 days
  TotalUsed        DECIMAL(5,2) NOT NULL DEFAULT 0,

  CONSTRAINT FK_Quota_Employee
    FOREIGN KEY (EmployeeId) REFERENCES Employees(EmployeeId),
  CONSTRAINT FK_Quota_LeaveType
    FOREIGN KEY (LeaveTypeId) REFERENCES LeaveTypes(LeaveTypeId),
  CONSTRAINT UQ_Employee_LeaveType_Year
    UNIQUE (EmployeeId, LeaveTypeId, LeaveYear)
);
```

- **Cascading dropdown 3 (Leave Type)**
  After Employee selection:
  - Show only those LeaveTypes where remaining balance > 0:
    TotalAllocated - TotalUsed > 0 for that EmployeeId and current LeaveYear.

  Example query for dropdown 3:

```
SELECT lt.LeaveTypeId, lt.LeaveTypeName
FROM EmployeeLeaveQuota q
JOIN LeaveTypes lt ON lt.LeaveTypeId = q.LeaveTypeId
WHERE q.EmployeeId = @EmployeeId
 AND q.LeaveYear  = @Year
 AND (q.TotalAllocated - q.TotalUsed) > 0
 AND lt.IsActive = 1;
```

---

5. LeaveApplications

Stores each leave request.

```
CREATE TABLE LeaveApplications (
  LeaveApplicationId INT PRIMARY KEY IDENTITY(1,1),
  EmployeeId       INT NOT NULL,
  LeaveTypeId      INT NOT NULL,
  FromDate         DATE NOT NULL,
  ToDate           DATE NOT NULL,
  TotalDays        DECIMAL(5,2) NOT NULL,
  Reason           VARCHAR(500) NOT NULL,
  Status           VARCHAR(20) NOT NULL DEFAULT 'Pending',
                   -- e.g., Pending, Approved, Rejected, Cancelled
```

```
    AppliedOn          DATETIME NOT NULL DEFAULT GETDATE(),
    ApprovedBy         INT NULL,          -- FK to Employees or Users table (optional)
    ApprovedOn         DATETIME NULL,

    CONSTRAINT FK_LeaveApp_Employee
        FOREIGN KEY (EmployeeId) REFERENCES Employees(EmployeeId),
    CONSTRAINT FK_LeaveApp_LeaveType
        FOREIGN KEY (LeaveTypeId) REFERENCES LeaveTypes(LeaveTypeId)
);
```

Overlapping Leave Check (logic)

Before inserting, check for overlaps for the same employee:

```
SELECT 1
FROM LeaveApplications
WHERE EmployeeId = @EmployeeId
  AND Status IN ('Pending', 'Approved')
  AND (
        (FromDate <= @ToDate AND ToDate >= @FromDate)
      );
```

If this returns a row, **block** the new request.

---

6. Optional: Holidays (if you want to exclude holidays)

```
CREATE TABLE Holidays (
    HolidayId     INT PRIMARY KEY IDENTITY(1,1),
    HolidayDate   DATE NOT NULL UNIQUE,
    Description   VARCHAR(200) NOT NULL
);
```

You can then adjust TotalDays calculation in the application layer (or via function) to exclude weekends/holidays