

## **Proposed Database Objects to Level Up Your Project:**

### **1. Views**

Views are ideal for simplifying complex reporting.

- **v\_employee\_full\_details**: A view that joins all tables to show: Full Name, Job Title, Department, City, Country, and Region.
- **v\_department\_stats**: A view showing for each department: total employees, sum of salaries, average salary, and the manager's name.
- **v\_management\_chain**: A recursive view (or one that simulates hierarchy) showing each employee along with their direct supervisor and their supervisor's boss.

### **2. Stored Procedures**

Use these to automate "write" processes or calculations.

- **sp\_promote\_employee**: A procedure that receives p\_employee\_id and p\_new\_job\_id. It must:
  - o Insert the current record into job\_history.
  - o Update job\_id and hire\_date in the employees table.
- **sp\_adjust\_department\_salary**: Receives a department\_id and a percentage (e.g., 0.10 for 10%). It increases the salary of all employees in that department but validates that the new salary does not exceed the max\_salary defined in the jobs table.
- **sp\_transfer\_employee**: Moves an employee to a new department and automatically updates their manager to the manager of the new department.

### **3. Functions**

Useful for calculations that return a single value.

- **fn\_get\_annual\_compensation**: Receives employee\_id and calculates annual compensation considering commission (\$Salary \times 12 \times (1 + commission)\$).
- **fn\_years\_in\_company**: Receives an employee\_id and returns how many years they have been working (based on hire\_date).

### **4. Triggers**

To maintain integrity and auditing.

- **tr\_check\_salary\_range**: A BEFORE INSERT or BEFORE UPDATE trigger on the employees table that verifies if the entered salary is within the range (min\_salary and max\_salary) of the jobs table. If not, it throws an error.

- **tr\_audit\_salary\_change:** Every time a salary is updated, the trigger must insert the old value, the new value, the user who made the change, and the timestamp into an audit table (which you must create).
- **tr\_job\_history\_auto\_fill:** When an employee's job\_id is changed, the trigger should automatically insert the corresponding row into job\_history.

## 5. Indexes

To optimize existing queries.

- **Composite index on employees(last\_name, first\_name):** For fast name searches.
- **Index on employees(salary):** To optimize salary range filters.
- **Index on locations(city):** Since city is a common filter but not a primary key.

## Advanced "Real-World" Ideas:

### 1. Analytics & Reporting (*Intelligence Views*)

- **v\_attrition\_risk:** A view to identify employees with "flight risk." For example: those who have been in the same position (job\_id) for more than 5 years without changes in job\_history and whose salary is less than 80% of their position's max\_salary.
- **v\_salary\_gap:** A view that compares each employee's salary against the average of their city or country, not just their department.
- **v\_company\_hierarchy:** A view that generates a "Path" (e.g., President > VP > Manager > Employee) using Recursive CTEs.

### 2. Complex Business Procedures

- **sp\_calculate\_payroll\_tax:** A procedure to calculate taxes based on salary brackets. It could receive the employee ID and return the net salary after applying country-specific tax rules.
- **sp\_reorganize\_department:** If a department closes, this procedure moves all employees to a new department and updates their managers in a single transaction.

### 3. Security and Auditing (*Control Triggers*)

- **tr\_prevent\_weekend\_updates:** A trigger that prevents modifying salaries or terminating employees on Saturdays and Sundays (common in banking/corporate systems).
- **tr\_enforce\_manager\_department:** A trigger that validates that a manager\_id assigned to an employee belongs to the same department (or a hierarchically superior one).

#### **4. Automation with Events (Event Scheduler)**

- **ev\_annual\_bonus\_reset**: An event that runs every January 1st to reset or calculate productivity bonuses based on employee seniority.
- **ev\_archive\_job\_history**: A monthly event that moves job\_history records older than 10 years to an "archive" table to keep the main table efficient.

#### **5. Model Extensions (New Tables)**

- **performance\_reviews**: Table for annual evaluations (1 to 5 stars).
- **benefits**: Table for benefits (Health insurance, Gym, etc.) and an intermediate employee\_benefits table for **Many-to-Many (N:M)** relationships.

### **Data Architect / Engineer Level: "The Crown Jewels"**

#### **1. Data Versioning (Slowly Changing Dimensions - SCD Type 2)**

- **Idea**: Create a salary\_history table and a trigger that, upon any salary change, saves the previous value, the new value, and the date range (valid\_from, valid\_to).
- **Pro Query**: "What was Employee X's salary exactly on May 15, 2024?"

#### **2. Soft Delete Implementation**

- **Idea**: Add an is\_active (BOOLEAN) column and a termination\_date to the employees table instead of using hard DELETE.
- **Security Views**: Create a v\_active\_employees view to automatically filter out inactive employees for application use.

#### **3. JSON Generation for APIs**

- **Idea**: Create a Stored Procedure that returns an employee's full profile in **JSON** format, including an array of their job\_history.

#### **4. Schema "Health Check" Dashboards**

- **Idea**: A procedure sp\_db\_health\_report to detect:
  - o Departments without employees.
  - o Employees earning more than their own manager (salary anomaly).
  - o Countries without any associated locations.

## Summary of Objects by Difficulty Level

Level	Object	Purpose
Basic	View v_payroll	Join tables to see names and net salaries.
Intermediate	Trigger tr_audit	Monitor changes to sensitive data.
Advanced	Recursive CTE	Draw the full organizational chart (hierarchical tree).
Expert	SCD Type 2	Maintain history of changes for any column over time.