horizontal line



# Project Name: Managing Employee Performance and Satisfaction data.

Member 1: SARASWATHY S.

Member 2: PRAVEENKUMAR.

# PROJECT OBJECTIVE:

The objective of this project is to design and implement a comprehensive SQL database system for managing employee performance and satisfaction data. This system will facilitate efficient data storage, retrieval, and analysis to support HR decision-making processes. The key goals include:

1. Data Integration: Integrate various data points related to employee demographics, education, performance ratings, and satisfaction levels into a unified database.
2. Performance Analysis: Enable detailed analysis of employee performance metrics, including job satisfaction, manager ratings, and training opportunities.
3. Satisfaction Tracking: Track and analyze employee satisfaction levels across different dimensions such as environment, job, and relationships.
4. Advanced Querying: Provide the capability to perform complex queries, aggregations, and joins to extract meaningful insights from the data.
5. Reporting: Generate reports that summarize key performance indicators and satisfaction metrics for different departments and roles.
6. Decision Support: Support HR and management in making informed decisions regarding promotions, training programs, and employee retention strategies.

## Key Features:

* Comprehensive Schema: A well-defined schema that includes tables for employee details, education levels, performance ratings, rating levels, and satisfaction levels.
* Data Integrity: Ensure data integrity through the use of primary and foreign keys, as well as appropriate data types and constraints.
* Advanced SQL Queries: Implement complex SQL queries to perform detailed analysis and generate insights.
* Stored Procedures and Functions: Utilize stored procedures and functions to automate common tasks and calculations.
* User-Friendly Interface: Optionally, develop a user-friendly interface for HR personnel to interact with the database and generate reports.

## Expected Outcomes:

* A fully functional SQL database that can store and manage employee-related data efficiently.
* The ability to perform advanced data analysis to uncover trends and insights.
* Enhanced decision-making capabilities for HR and management through detailed reports and data visualizations.

By achieving these objectives, the project will provide a robust foundation for managing and analyzing employee performance and satisfaction data, ultimately contributing to improved organizational effectiveness and employee well-being.

# SELECTING A DATASET:

Step-by-Step Guide to Select a Dataset from Kaggle.

1. Visit Kaggle Website:
   * Go to [Kaggle](https://www.kaggle.com/).
2. Sign In/Sign Up:
   * If you already have an account, click on “Sign In” and enter your credentials.
   * If you don’t have an account, click on “Sign Up” and create a new account.
3. Navigate to Datasets:
   * On the Kaggle homepage, click on the “Datasets” tab located at the top of the page.
4. Search for a Dataset:
   * Use the search bar to enter keywords related to the dataset you are looking for.
   * You can also browse through categories or tags to find datasets that match your interests.
5. Filter Results:
   * Use the filters on the left side of the page to narrow down your search. You can filter by file type, size, license, and more.
6. Select a Dataset:
   * Click on a dataset that interests you to view more details. This will take you to the dataset’s page where you can see a description, files, and other relevant information.
7. Explore the Dataset:
   * Review the dataset’s description, columns, and sample data to ensure it meets your needs.
   * Check the “Discussion” and “Kernels” tabs for additional insights and analyses done by other users.
8. Download the Dataset:
   * Click on the “Download” button to download the dataset to your local machine. The dataset will typically be in a ZIP file format.
9. Unzip the File:
   * Extract the contents of the ZIP file to access the dataset files.
10. Load the Dataset:
    * Use your preferred data analysis tool (e.g., Jupyter Notebook, RStudio, Excel) to load and start working with the dataset.

# DATA EXPLORATION:

Data exploration is a crucial first step in the data analysis process. It involves examining your dataset to understand its structure, identify patterns, detect anomalies, and gain insights. Here are the key steps and techniques involved in data exploration:

Steps in Data Exploration:

1. Data Collection:
   * Gather data from various sources such as databases, APIs, or CSV files.
   * Ensure the data is relevant to your analysis objectives.
2. Data Cleaning:
   * Handle missing values by either removing or imputing them.
   * Identify and correct inconsistencies and errors in the data.
   * Remove duplicates to ensure data integrity.
3. Data Profiling:
   * Generate summary statistics (mean, median, mode, standard deviation) for numerical data.
   * Understand the distribution of categorical data by calculating frequency counts.
   * Identify outliers and unusual patterns.
4. Data Visualization:
   * Use plots and charts (histograms, box plots, scatter plots) to visualize data distributions and relationships.
   * Create correlation matrices to understand relationships between variables.
5. Feature Engineering:
   * Create new features from existing data to improve model performance.
   * Normalize and scale numerical features to ensure consistency.
   * Encode categorical variables using techniques like one-hot encoding.
6. Exploratory Data Analysis (EDA):
   * Perform in-depth analysis to uncover patterns and trends.
   * Use statistical tests to validate hypotheses and understand data relationships.

## Techniques in Data Exploration:

1. Descriptive Statistics:
   * Calculate measures of central tendency (mean, median) and dispersion (variance, standard deviation).
2. Data Visualization:
   * Use visual tools like histograms, bar charts, line graphs, and scatter plots to explore data visually.
3. Correlation Analysis:
   * Calculate correlation coefficients to understand the strength and direction of relationships between variables.
4. Outlier Detection:
   * Identify and analyze outliers using box plots and z-scores.
5. Dimensionality Reduction:
   * Use techniques like Principal Component Analysis (PCA) to reduce the number of features while retaining important information.

## Benefits of Data Exploration:

* Improved Data Quality: Identifying and correcting errors ensures the reliability of your analysis.
* Insight Discovery: Uncover hidden patterns and relationships in the data.
* Informed Decision-Making: Provides a solid foundation for building predictive models and making data-driven decisions.

# DATABASE SETUP:

Step 1: Download the Dataset from Kaggle.

1. Visit Kaggle: Go to Kaggle.
2. Sign In/Sign Up: Log in to your account or create a new one.
3. Search for a Dataset: Use the search bar to find a dataset relevant to your project.
4. Download the Dataset: Click on the dataset you want, then click the “Download” button. The dataset will be downloaded as a ZIP file.

Step 2: Extract the Dataset.

1. Unzip the File: Extract the contents of the ZIP file to access the dataset files, usually in CSV format.

Step 3: Set Up Your Database.

1. Install MySQL: If you haven’t already, download and install MySQL from the official website.
2. Start MySQL Server: Open MySQL Workbench.

Step 4: Import the Dataset.

1. Use the Table Data Import Wizard:
   * Open MySQL Workbench and connect to your database.
   * Click on the “Table Data Import Wizard” button in the “Navigator” panel.
   * Select “CSV Data File” as the data source and click “Next”.
   * Select the downloaded CSV file and click “Open”.
   * Configure the import settings, such as the delimiter, enclosure, and text qualifier, and click “Next”.
   * Select the target table or create a new one, and map the CSV columns to the table columns.
   * Click “Next” and then “Start Import”.

Step 5: Verify the Data Import.

1. Run Queries: Test your setup by running some queries to ensure the data has been imported correctly.

# SCHEMA DESIGN:

## Table: EducationLevel

CREATE TABLE EducationLevel (

EducationLevelId INT PRIMARY KEY,

EducationalLevel VARCHAR(255) NOT NULL );

## Table: PerformanceRating

CREATE TABLE PerformanceRating (

PerformanceId VARCHAR(50) PRIMARY KEY,

EmployeeId VARCHAR(50) NOT NULL,

ReviewDate DATE NOT NULL,

EnvironmentSatisfaction INT,

JobSatisfaction INT,

RelationshipSatisfaction INT,

TrainingOppurtunitiesWithinYr INT,

TrainingOppurtunitiesTaken INT,

WorkLifeBalance INT,

SelfRating INT,

ManagerRating INT,

FOREIGN KEY (EmployeeId) REFERENCES Employee(EmployeeId)

);

## Table: RatingLevel

CREATE TABLE RatingLevel (

RatingId INT PRIMARY KEY,

RatingLevel VARCHAR(255) NOT NULL

);

## Table: SatisfiedLevel

CREATE TABLE SatisfiedLevel (

SatisfactionId INT PRIMARY KEY,

SatisfactionLevel VARCHAR(255) NOT NULL

);

## Table: Employee

CREATE TABLE Employee (

EmployeeId VARCHAR(50) PRIMARY KEY,

FirstName VARCHAR(255) NOT NULL,

LastName VARCHAR(255) NOT NULL,

Gender VARCHAR(10),

Age INT,

BusinessTravel VARCHAR(255),

Department VARCHAR(255),

DistanceFromHome INT,

State VARCHAR(255),

Ethnicity VARCHAR(255),

Education INT,

EducationField VARCHAR(255),

JobRole VARCHAR(255),

MaritalStatus VARCHAR(255),

Salary INT,

StockOptionLevel INT,

OverTime CHAR(1),

HireDate DATE,

Attrition CHAR(1),

YearsAtCompany INT,

YearsInMostRecentRole INT,

YearsSinceLastPromotions INT,

YearsWithCurrManager INT,

FOREIGN KEY (Education) REFERENCES EducationLevel(EducationLevelId)

);

This schema includes the following:

* **Primary Keys:** Each table has a primary key to uniquely identify each record.
* **Foreign Keys:** The PerformanceRating table references the employee table, and the employee table references the EducationLevel table to maintain referential integrity.
* **Data Types:** Appropriate data types are used for each column based on the provided information.

# QUERY DEVELOPMENT:

1. Write a SQL command to count the number of employees in each education level.

***SELECT el.EducationalLevel, COUNT(e.EmployeeId) AS Employee\_Count***

***FROM Employee e***

***JOIN EducationLevel el ON e.Education = el.EducationLevelId***

***GROUP BY el.EducationalLevel;***

1. Write a SQL command to find employees who have taken more than 2 training opportunities within a year.

***SELECT e.FirstName, e.LastName, p.TrainingOpportunitiesTaken***

***FROM Employee e***

***JOIN PerformanceRating p ON e.EmployeeId = p.EmployeeId***

***WHERE p.TrainingOpportunitiesTaken > 2;***

1. Write a SQL command to List employees who have been with the company for more than 5 years.

***SELECT FirstName, LastName, YearsAtCompany***

***FROM Employee***

***WHERE YearsAtCompany > 5;***

1. Write a SQL command to calculate the total number of training opportunities taken by all employees.

***SELECT SUM(TrainingOpportunitiesTaken) AS Total\_Training\_Taken***

***FROM PerformanceRating;***

1. Write a SQL command to find the top 3 highest-paid employees in each department.

***SELECT Department, FirstName, LastName, Salary***

***FROM (***

***SELECT Department, FirstName, LastName, Salary,***

***ROW\_NUMBER() OVER (PARTITION BY Department ORDER BY Salary DESC***

***) AS rank FROM Employee) AS ranked***

***WHERE rank <= 3;***

1. Write a SQL command to calculate the cumulative salary for each department.

***SELECT Department, FirstName, LastName, Salary,***

***SUM(Salary) OVER (PARTITION BY Department ORDER BY Salary DESC) AS Cumulative\_Salary***

***FROM Employee;***

1. Write a SQL command to find employees who have a higher salary than the average salary of their department.

***SELECT e.FirstName, e.LastName, e.Salary, e.Department***

***FROM Employee e***

***JOIN (***

***SELECT Department, AVG(Salary) AS avg\_salary***

***FROM Employee***

***GROUP BY Department***

***) AS Dept\_Avg ON e.Department = Dept\_Avg.Department***

***WHERE e.Salary > Dept\_Avg.Avg\_Salary;***

1. Write a SQL command to calculate the average job satisfaction for each department, considering only employees with more than 5 years at the company.

***SELECT e.Department, AVG(p.JobSatisfaction) AS Avg\_JobSatisfaction***

***FROM Employee e***

***JOIN PerformanceRating p ON e.EmployeeId = p.EmployeeId***

***WHERE e.YearsAtCompany > 5***

***GROUP BY e.Department;***

1. Write a SQL command to list employees along with their satisfaction levels.

***SELECT e.FirstName, e.LastName, s.SatisfactionLevel***

***FROM Employee e***

***JOIN SatisfiedLevel ON e.EmployeeId = s.SatisfactionId;***

1. Write a SQL command to find the average age of employees in each department.

***SELECT Department, AVG(Age) AS Average\_Age***

***FROM Employee***

***GROUP BY Department;***

1. Write a SQL command to list employees who have the highest job satisfaction in their department.

***SELECT e.FirstName, e.LastName, e.Department, p.JobSatisfaction***

***FROM Employee e***

***JOIN PerformanceRating p ON e.EmployeeId = p.EmployeeId***

***WHERE p.JobSatisfaction = (***

***SELECT MAX(p2.JobSatisfaction)***

***FROM PerformanceRating p2***

***JOIN Employee e2 ON p2.EmployeeId = e2.EmployeeId***

***WHERE e2.Department = e.Department***

***);***

1. Write a SQL command to calculate the average self-rating for employees who have been with the company for more than 10 years.

***SELECT AVG(p.SelfRating) AS Avg\_Self\_Rating***

***FROM Employee e***

***JOIN PerformanceRating p ON e.EmployeeId = p.EmployeeId***

***WHERE e.YearsAtCompany > 10;***

1. Write a query to list all employees who share the same manager. Use a self join on the Employee table to find employees with the same YearsWithCurrManager.

***SELECT e1.FirstName AS Employee1, e2.FirstName AS Employee2, e1.YearsWithCurrManager***

***FROM Employee e1***

***INNER JOIN Employee e2 ON e1.YearsWithCurrManager = e2.YearsWithCurrManager***

***WHERE e1.EmployeeID != e2.EmployeeID;***

1. Write a query to join Performance Rating and Satisfied Level tables on Environment Satisfaction and display the employees who have both Environment Satisfaction of 4 and above, and Work Life Balance of 4.

***SELECT p.EmployeeID, p.EnvironmentSatisfaction, p.WorkLifeBalance***

***FROM PerformanceRating p***

***JOIN SatisfiedLevel s***

***ON p.EnvironmentSatisfaction = s.SatisfactionID***

***WHERE p.EnvironmentSatisfaction >= 4 AND p.WorkLifeBalance = 4;***

1. Write a query using a left outer join to display all education levels from the Education Level table and their corresponding Employee IDfrom the Performance Rating table, if available.

***SELECT e.EducationLevel, p.EmployeeID, p.JobSatisfaction***

***FROM EducationLevel e***

***LEFT JOIN PerformanceRating p***

***ON e.EducationLevelID = p.EmployeeID;***

1. Write a stored procedure that calculates the average Job Satisfaction for all employees from the Performance Rating table.

***delimiter //***

***CREATE PROCEDURE Average ()***

***BEGIN***

***SELECT AVG(JobSatisfaction) AS AvgJobSatisfaction***

***FROM PerformanceRating;***

***END;***

***//***

***delimiter;***

1. Write a stored procedure that accepts an Employee ID and updates the salary for that employee in the Employee table.

***CREATE PROCEDURE UpdateEmployeeSalary(IN EmpID VARCHAR(10), IN***

***NewSalary DECIMAL(10, 2))***

***BEGIN***

***UPDATE Employee***

***SET Salary = NewSalary***

***WHERE EmployeeID = EmpID;***

***END;***

1. Create a function that takes an employee’s Employee ID as input and returns their overall performance score as the sum of their Self Rating and Manager Rating from the Performance Rating table.

***CREATE FUNCTION GetOverallPerformanceScore(EmpID VARCHAR(10))***

***RETURNS INT***

***BEGIN***

***DECLARE overall\_score INT;***

***SELECT (SelfRating + ManagerRating) INTO overall\_score***

***FROM PerformanceRating***

***WHERE EmployeeID = EmpID;***

***RETURN overall\_score;***

***END;***

1. Write a query using a CTE to retrieve the First Name, Last Name, and Years At Company for all employees who have worked more than 5 years.

***WITH Emp AS (***

***SELECT FirstName, LastName, YearsAtCompany***

***FROM Employee***

***WHERE YearsAtCompany > 5***

***)***

***SELECT \* FROM Emp;***

1. List employees who have the same job role as their manager.

***SELECT e.FirstName, e.LastName, e.JobRole***

***FROM Employee e***

***JOIN Employee m ON e.ManagerId = m.EmployeeId***

***WHERE e.JobRole = m.JobRole;***

# CONCLUSION:

The successful design and implementation of the comprehensive SQL database system for managing employee performance and satisfaction data have significantly enhanced the HR department’s ability to make informed decisions.