**Employee Management System**

This will help you create a database for an Employee Management System (EMS). The EMS database will assist in managing employee details, department information, roles, salary records, and performance feedback.

The Employee Management System database will consist of the following tables:

1. Employee: Stores information about individual employees.
2. Department: Contains details about different departments within the organization.
3. Project Details: Track information related to various projects in the organization, including project names, budgets, and locations.
4. Salary: Tracks salary information for employees.

We will use MySQL Workbench as the DBMS to create the database and its related operations.

1. **Introduction to MySQL Workbench**

What is MySQL?

MySQL is an open-source Relational Database Management System (RDBMS) developed by Oracle Corporation, Sun Microsystems, that uses [Structured Query language(SQL)](https://www.simplilearn.com/tutorials/sql-tutorial/what-is-sql) to interact with databases. You can use MySQL to store, retrieve, manipulate and process data that is in the form of tables.

Why Use MySQL?

There are various relational database management systems present in the tech world today, such as Microsoft SQL Server, Microsoft Access, Oracle, DB2, etc.

Here are some reasons why people use MySQL over other Database Management Systems.

* Multiple Storage Engines

MySQL adheres to multiple storage engines, and each one of the storage engines possesses unique features, while other databases like SQL Server only support a single storage engine.

* InnoDB: It is the default storage engine fabricated with MySQL since version 5.5. It supports ACID-based transactions.
* MyISAM: Former to version 5.5, MyISAM was the default storage engine used by MySQL. It does not support ACID-based transactions.
* High Performance

MySQL has reported high performance compared to other database management systems because of its simplicity in design, and adherence to multiple storage engines.

* Cost-Effective

The community edition of MySQL is free of cost, and the commercial edition has a licensing fee, which is cost-effective compared to other products available in the market.

* Cross-Platform

MySQL runs on Windows, Linux, and macOS because of its cross-platform property.

With this, it is clear why MySQL is used. Now, you will see what MySQL Workbench exactly is and how one can use it.

What is MySQL Workbench?

MySQL Workbench is a unified cross-platform, open-source relational database design tool that adds functionality and ease to MySQL and SQL development work. MySQL Workbench provides data modeling, SQL development, and various administration tools for configuration. It also offers a graphical interface to work with the databases in a structured way.

* You can create a Graphical Model using MySQL Workbench
* MySQL Workbench provides reverse engineering for live databases to models
* MySQL Workbench offers a forward engineering model to a script/live database

1. **Installation of MySQL Workbench**

The installation process is similar to other operating systems.

1. Open the MySQL website on a browser. Click on the following link: [MySQL Downloads](https://www.mysql.com/downloads/).
2. Select the Downloads option.
3. Select MySQL Installer for Windows.
4. Choose the desired installer and click on download.
5. After it downloads the installer, open it.
6. It will ask for permission; when it does, click Yes. The installer will then open. Now, it will ask to choose the setup type, here, select Custom.
7. Click on Next. With this, you will be able to install MySQL server, MySQL Workbench, and MySQL shell.
8. Open MySQL Servers, select the server you want to install, and move it to the  Products/Features to be installed window section. Now, expand Applications, choose MySQL Workbench and MySQL shell. Move both of them to ‘Products/Features to be installed’.
9. Click on the Next button. Now, click on the Execute button to download and install the MySQL server, MySQL Workbench, and the MySQL shell.
10. Once the product is ready to configure, click on Next. Under Type and Networking, go with the default settings and select Next.
11. For authentication, use the recommended strong password encryption.
12. Set your MySQL Root password and click on next.
13. Go for the default windows service settings and under apply configuration, click on execute. Once the configuration is complete, click on finish.
14. Complete the installation. This will now launch the MySQL Workbench and the MySQL Shell.

Once MySQL Workbench is installed, select the Local instance and enter the password.

Now, you can use the MySQL query tab to write your SQL queries.

**Identify Entities**

* Start by identifying the main entities in your system. These are the objects or concepts about which you want to store data.
* Each entity should correspond to a table in your database.

**Define Attributes**

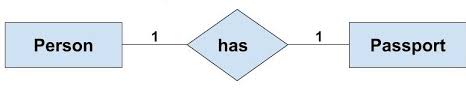
* For each entity, list the attributes (properties or fields) that describe it.
* These attributes will become columns in the corresponding database table.

**Identify Relationships**

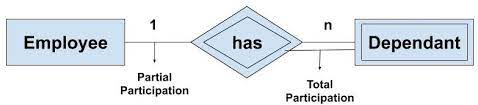
* Determine how entities are related to each other. There are three types of relationships: one-to-one (1:1), one-to-many (1:N), and many-to-many (N:M).
* Represent these relationships using lines connecting the entities.

Let’s see a few examples of relationships:

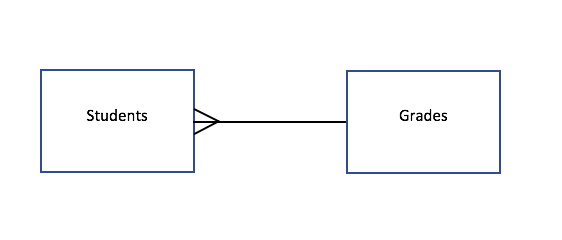
1. **One-to-One:** When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.



1. **One-to-Many:** In one-to-many mapping as well where each entity can be related to more than one relationship and the total number of tables that can be used in this is 2. Let us assume that one surgeon deparment can accomodate many doctors. So the Cardinality will be 1 to M. It means one deparment has many Doctors.



**3. Many-to-One:** When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students.



**Many-to-Many:**When entities in all entity sets can take part more than once in the relationship cardinality is many to many.



**Cardinality Notation**

Cardinality represents the number of times an entity of an entity set participates in a relationship set. Or we can say that the cardinality of a relationship is the number of tuples (rows) in a relationship.

* Use notation (such as Crow's Foot Notation or Chen Notation) to indicate the cardinality of each relationship.
* Cardinality describes how many instances of one entity are related to how many instances of another entity.
* Common notations include:
* One (1)
* Zero or one (0..1)
* Many (N)
* Zero or many (0..N)

**Optional:**

**Add Attributes and Constraints**

* Include additional information in your ERD, such as primary keys, foreign keys, and constraints (e.g., unique constraints).

**Create the Diagram**

* Use specialized diagramming software or tools (e.g., Lucidchart, draw.io, or even pen and paper) to create your ERD.

**Refine and Review:**

* Review your ERD with stakeholders and team members to ensure it accurately represents the data model and relationships. Make any necessary refinements.
* Entities of Employee Management System

1. Employee
2. Department
3. Salary
4. Project

* let’s identify the attributes and relationships of each entity for the Employee Management System.
* Employee

Attributes:

1. **EmployeeID** (Primary Key):

An auto-incremented unique identifier for each employee.

1. **First\_name**:

The first name of the employee.

1. **Last\_name**:

The last name of the employee.

1. **Address**:

The address or physical location of the employee.

1. **Salary**:

The salary of the employee.

1. **Gender**:

The gender of the employee.

1. **DepartmentID** (Foreign Key):

A reference to the department to which the employee belongs, establishing a many-to-one relationship with the "Department" table.

**Relationships:**

Many **Employees** can enroll in one **Department**(Many-to-One)

* Department

Attributes:

1. **DepartmentID** (Primary Key):

An auto-incremented unique identifier for each department.

1. **Name**:

The name of the department.

Relationshiops:

One Department can belongs to Many Employee.

* Salary

Attributes:

1. **SalaryID** (Primary Key):

An auto-incremented unique identifier for each salary record.

1. **EmployeeID** (Foreign Key):

A reference to the employee for whom the salary record is maintained, establishing a one-to-many relationship with the "Employee" table.

1. **SalaryAmount**:

The amount of the salary for a specific period.

1. **StartDate**:

The start date of the salary period.

1. **EndDate**:

The end date of the salary period.

* Project

Attributes:

1. **ProjectID** (Primary Key):

An auto-incremented unique identifier for each project.

1. **Name**:

The name of the project.

1. **Budget**:

The budget allocated to the project.

1. **Location**:

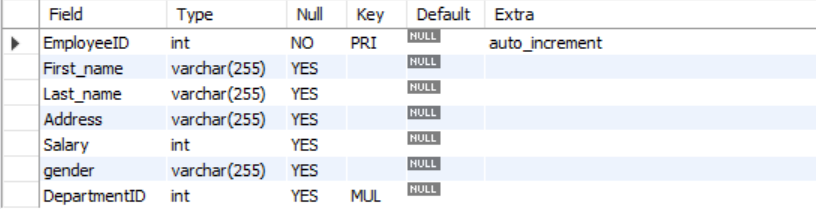
The location or place where the project is based.

Relationship:

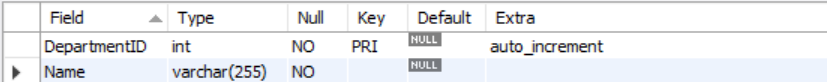
Many Projects can assign to Many Employees.

**Table Structure**

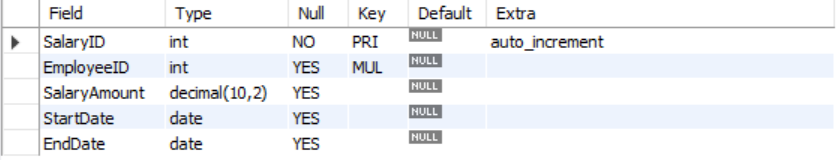
* 1. *Employee*



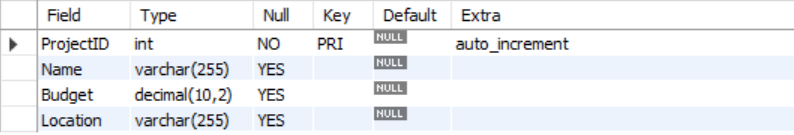
* 1. *Department*



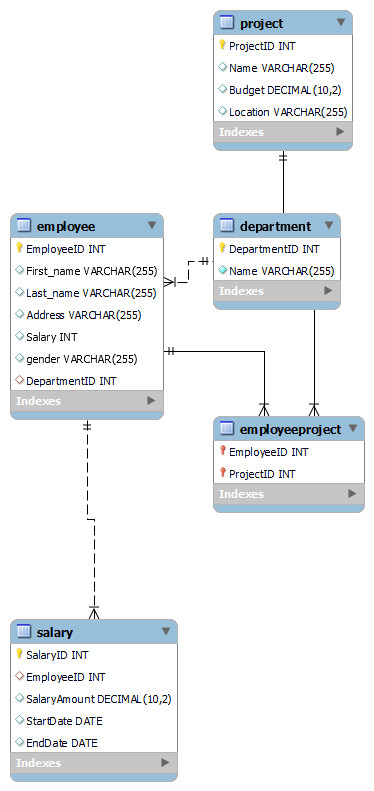
* 1. *Salary*



* 1. *Project*



* ERD Diagram



**In this Entity-Relationship Diagram:**

* **Employees can be assigned to projects**, and each project can have multiple employees assigned to it, creating a many-to-many relationship.
  + The "EmployeeProject" entity serves as a bridge table between "Employee" and "Project" entities to represent this relationship.
* **Multiple employees can belong to one department** (many-to-one relationship).
* **Each department can have multiple employees** (one-to-many relationship).
* **Each employee can have multiple salary records** to track their salary history.
* **Each employee can be assigned to manage multiple employees** (self-referencing relationship).

**1 . Creating a Database**

Using MySQL Workbech, create a new database for your Employee management system.

Query:

create database employeedb;

2. Using a database

Query:

use employeedb;

3. Creating the Tables for each entity

Query:

CREATE TABLE Employee (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

First\_name VARCHAR(255),

Last\_name VARCHAR(255),

Address VARCHAR(255),

Salary INT(10),

gender VARCHAR(255),

DepartmentID INT,

FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)

);

CREATE TABLE department (

DepartmentID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(255) NOT NULL

);

CREATE TABLE Salary (

SalaryID INT PRIMARY KEY AUTO\_INCREMENT,

EmployeeID INT,

SalaryAmount DECIMAL(10, 2),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID)

);

CREATE TABLE Project (

ProjectID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(255),

Budget DECIMAL(10, 2),

Location VARCHAR(255)

);

4. **Insert Records**

Query:

INSERT INTO Employee (First\_name, Last\_name, Address, Salary, gender, DepartmentID)

VALUES

('Raj', 'Gupta', 'Mumbai', 60000, 'Male', 1),

('Deepika', 'Patil', 'Pune', 75000, 'Female', 1),

('Priya', 'Sharma', 'Nashik', 80000, 'Female', 2),

('Vaishnavi', 'Shinde', 'Ahemdnagar', 90000, 'Female', 2);

INSERT INTO Department (Name)

VALUES

('HR'),

('Finance'),

('Engineering'),

('Marketing');

INSERT INTO Salary (EmployeeID, SalaryAmount, StartDate, EndDate)

VALUES

(1, 60000.00, '2023-01-01', '2023-12-31'),

(2, 75000.00, '2023-01-01', '2023-12-31'),

(3, 80000.00, '2023-01-01', '2023-12-31');

INSERT INTO Project (Name, Budget, Location)

VALUES

('Project A', 100000.00, 'New York'),

('Project B', 75000.00, 'Los Angeles'),

('Project C', 120000.00, 'San Francisco');

5. **Select Records**

Query:

Select \* From Employee;

1. Retrieve data for employees who belong to a specific department (DepartmentID = 1):

SELECT \*

FROM Employee

WHERE DepartmentID = 1;

1. Retrieve employees with a salary greater than 70,000:

SELECT \*

FROM Employee

WHERE Salary > 70000;

1. Retrieve employees in a specific department (e.g., DepartmentID = 2) with a salary greater than 80,000:

SELECT \*

FROM Employee

WHERE DepartmentID = 2 AND Salary > 80000;

1. Retrieve employees whose last names start with "S":

SELECT \*

FROM Employee

WHERE Last\_name LIKE 'S%';

1. Retrieve employees who live in a specific city (e.g., "Mumbai"):

SELECT \*

FROM Employee

WHERE Address LIKE '%Mumbai%';

5. **Update Records**

Write SQL statements to update record(s) when needed. For example:

1. update the salary of the employee with an **EmployeeID** of 1 sql

Query:

UPDATE Employee

SET Salary = 65000

WHERE EmployeeID = 1;

1. Increase the salary of all employees in DepartmentID 1 by 10%)

Query:

UPDATE Employee

SET Salary = Salary \* 1.10

WHERE DepartmentID = 1;

1. Change the address of the employee with **EmployeeID** 1)

Query:

UPDATE Employee

SET Address = 'Pune

WHERE EmployeeID = 1;

6**. Delete Records**

1. delete the employee with **EmployeeID** 3

Query:

DELETE FROM Employee

WHERE EmployeeID = 3;

1. delete employees in a specific department

Query:

DELETE FROM Employee

WHERE DepartmentID = 2;

1. Delete employees with a salary less than 50000 or greater than 90000

Query:

DELETE FROM Employee

WHERE Salary < 50000 OR Salary > 90000;

7. Some other Queries:

* **JOIN**: Combines rows from two or more tables based on a related column.

Query:

SELECT Employee.First\_name, Department.Name

FROM Employee

JOIN Department ON Employee.DepartmentID = Department.DepartmentID;

* **GROUP BY**: Groups rows that have the same values into summary rows.

Query:

SELECT DepartmentID, AVG(Salary) AS AvgSalary

FROM Employee

GROUP BY DepartmentID;

* **ORDER BY**: Sorts the result set in ascending or descending order.

Query:

SELECT \* FROM Employee

ORDER BY Last\_name ASC;

* **COUNT**: Counts the number of rows in a result set.

Query:

SELECT COUNT(\*) FROM Employee;

* **SUM**: Calculates the sum of a numeric column.

Query:

SELECT SUM(Salary) FROM Employee;

* **AVG**: Calculates the average of a numeric column.

Query:

SELECT AVG(Salary) FROM Employee;

* **MAX** and **MIN**: Retrieves the maximum and minimum values from a column.

Query:

SELECT MAX(Salary) FROM Employee;

SELECT MIN(Salary) FROM Employee;s

* **DISTINCT Operator**: retrieve a list of distinct values for the "gender" column from the "Employee" table

Query:

SELECT DISTINCT gender

FROM Employee;

* **LIKE Operator**: retrieve employees whose last names start with "S,"

Query

SELECT \*

FROM Employee

WHERE Last\_name LIKE 'S%';: