Project Title: MySQL Employee Database: Tracking Sales and Relationships

Introduction

The MySQL Employee Database project is a comprehensive initiative designed to enhance organizational efficiency through the creation of a robust system for managing employee information, tracking sales, and handling relationships within the corporate structure. In today's dynamic business environment, effective employee management and insightful sales tracking are critical components for the success of any organization. This project addresses these needs by leveraging the power of MySQL, a widely used relational database management system.

Background:

The necessity for an organized and efficient employee data management system stem from the complexities of modern workplaces. As organizations grow, the volume of employee information increases, making it crucial to have a centralized database that ensures accurate and easily accessible data. Additionally, the project recognizes the significance of tracking sales and managing client relationships, essential aspects for businesses to thrive in competitive markets.

Objectives:

The primary objectives of the MySQL Employee Database project are twofold: to streamline the management of employee data and to facilitate the tracking of sales and client relationships. By achieving these objectives, the project aims to provide organizations with a reliable and scalable solution for optimizing their internal processes.

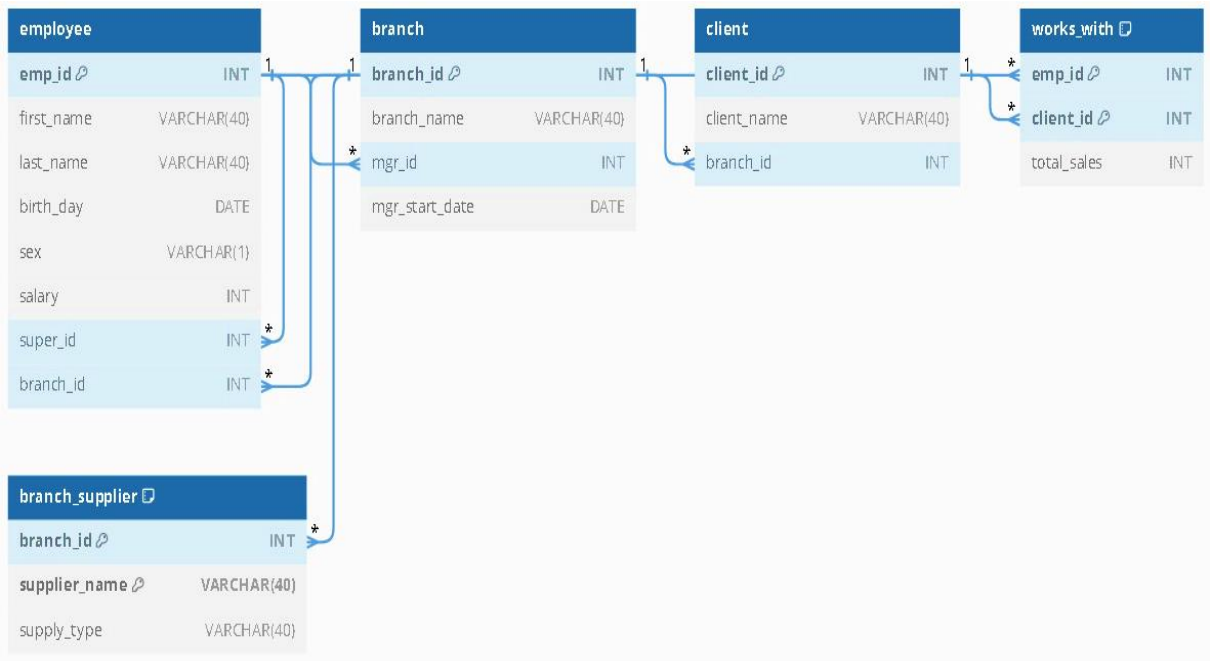
1. **Efficiently Manage Employee Data:** The project seeks to simplify the process of managing employee information. This

includes storing and retrieving employee details, such as names, birthdates, and branch assignments, in a structured and organized manner. By centralizing this information, the system aims to eliminate redundancy and improve the overall efficiency of HR processes.

2. **Track Sales and Client Relationships:** An integral aspect of the project is the ability to track sales and manage client relationships. Through the Works_With table, the system connects employees with clients, recording total sales for each interaction. This functionality provides valuable insights into the organization's revenue streams and client engagement, contributing to informed decision-making.
3. **Analyze Branch Performance:** The project incorporates features for analyzing branch performance based on sales data. By aggregating and summarizing sales information at the branch level, organizations can gain a comprehensive understanding of their overall performance and identify areas for improvement.

Entity-Relationship Diagram (ERD)

The Entity-Relationship Diagram (ERD) serves as the foundational blueprint for the MySQL Employee Database project, illustrating the relationships between key entities and their attributes. The ERD showcases the interconnectedness of entities, facilitating a visual understanding of the database's structure.



The main entities depicted in the ERD include:

- **Employee:** Represents individual employees with attributes such as emp_id, first_name, last_name, birth_day, sex, salary, super_id, and branch_id.
- **Branch:** Captures information about different branches within the organization, with attributes like branch_id, branch_name, mgr_id, and mgr_start_date.
- **Client:** Encompasses client details with attributes client_id, client_name, and branch_id.
- **Works_With:** Establishes the relationship between employees and clients, tracking total_sales as a measure of their interaction.

- **Branch_Supplier:** Represents suppliers associated with specific branches, with attributes branch_id, supplier_name, and supply_type.

Database Schema

Employee Table

The Employee table serves as a core component, storing comprehensive information about each employee. Attributes include emp_id as the primary key, along with first_name, last_name, birth_day, sex, salary, super_id (indicating the supervisor), and branch_id (denoting the employee's branch assignment).

Branch Table

The Branch table encapsulates data related to organizational branches. It includes attributes such as branch_id (primary key), branch_name, mgr_id (identifying the branch manager), and mgr_start_date.

Client Table

The Client table manages client-specific details, including client_id as the primary key, client_name, and branch_id (indicating the associated branch).

Works_With Table

The Works_With table establishes the relationship between employees and clients, facilitating the tracking of total sales. It includes emp_id and client_id as composite primary keys, along with the total_sales attribute.

Branch_Supplier Table

The Branch_Supplier table manages supplier information associated with specific branches. It consists of branch_id and supplier_name as composite primary keys, along with the supply_type attribute.

Implementation

Database Setup

The project is exclusively implemented using MySQL, a powerful relational database management system. The database setup involves creating tables to store employee, branch, client, works_with, and branch_supplier information. Primary keys and foreign keys are defined to establish relationships between tables and ensure data integrity.

Code:

```
CREATE TABLE employee (
  emp_id INT PRIMARY KEY,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  birth_day DATE,
  sex VARCHAR(1),
  salary INT,
  super_id INT,
  branch_id INT
);
CREATE TABLE branch (
  branch_id INT PRIMARY KEY,
  branch_name VARCHAR(40),
  mgr_id INT,
  mgr_start_date DATE,
  FOREIGN KEY(mgr_id) REFERENCES employee(emp_id) ON DELETE SET NULL
```

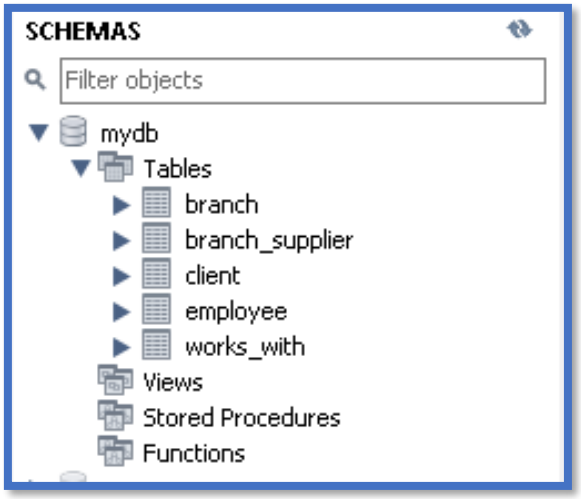
```
);
ALTER TABLE employee
ADD FOREIGN KEY(branch_id)
REFERENCES branch(branch_id)
ON DELETE SET NULL;

ALTER TABLE employee
ADD FOREIGN KEY(super_id)
REFERENCES employee(emp_id)
ON DELETE SET NULL;

CREATE TABLE client (
  client_id INT PRIMARY KEY,
  client_name VARCHAR(40),
  branch_id INT,
  FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE SET NULL
);
CREATE TABLE works_with (
  emp_id INT,
  client_id INT,
  total_sales INT,
  PRIMARY KEY(emp_id, client_id), FOREIGN KEY(emp_id) REFERENCES employee(emp_id) ON DELETE CASCADE,
```

```
  FOREIGN KEY(client_id) REFERENCES client(client_id) ON DELETE CASCADE
);
CREATE TABLE branch_supplier (
  branch_id INT,
  supplier_name VARCHAR(40),
  supply_type VARCHAR(40),
  PRIMARY KEY(branch_id, supplier_name),
  FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE CASCADE
);
-- ---------------------------------------------------------------------------
-- Corporate
INSERT INTO employee VALUES(100, 'David', 'Wallace', '1967-11-17', 'M', 250000, NULL, NULL);

INSERT INTO branch VALUES(1, 'Corporate', 100, '2006-02-09');

UPDATE employee
SET branch_id = 1
WHERE emp_id = 100;

INSERT INTO employee VALUES(101, 'Jan', 'Levinson', '1961-05-11', 'F', 110000, 100, 1);
```

```
-- Scranton
INSERT INTO employee VALUES(102, 'Michael', 'Scott', '1964-03-15', 'M', 75000, 100, NULL);

INSERT INTO branch VALUES(2, 'Scranton', 102, '1992-04-06');

UPDATE employee
SET branch_id = 2
WHERE emp_id = 102;

INSERT INTO employee VALUES(103, 'Angela', 'Martin', '1971-06-25', 'F', 63000, 102, 2);
INSERT INTO employee VALUES(104, 'Kelly', 'Kapoor', '1980-02-05', 'F', 55000, 102, 2);
INSERT INTO employee VALUES(105, 'Stanley', 'Hudson', '1958-02-19', 'M', 69000, 102, 2);

-- Stamford
INSERT INTO employee VALUES(106, 'Josh', 'Porter', '1969-09-05', 'M', 78000, 100, NULL);

INSERT INTO branch VALUES(3, 'Stamford', 106, '1998-02-13');

UPDATE employee
SET branch_id = 3
```

```
WHERE emp_id = 106;

INSERT INTO employee VALUES(107, 'Andy', 'Bernard', '1973-07-22', 'M', 65000, 106, 3);

INSERT INTO employee VALUES(108, 'Jim', 'Halpert', '1978-10-01', 'M', 71000, 106, 3);


-- BRANCH SUPPLIER

INSERT INTO branch_supplier VALUES(2, 'Hammer Mill', 'Paper');

INSERT INTO branch_supplier VALUES(2, 'Uni-ball', 'Writing Utensils');

INSERT INTO branch_supplier VALUES(3, 'Patriot Paper', 'Paper');

INSERT INTO branch_supplier VALUES(2, 'J.T. Forms & Labels', 'Custom Forms');

INSERT INTO branch_supplier VALUES(3, 'Uni-ball', 'Writing Utensils');

INSERT INTO branch_supplier VALUES(3, 'Hammer Mill', 'Paper');

INSERT INTO branch_supplier VALUES(3, 'Stamford Lables', 'Custom Forms');


-- CLIENT

INSERT INTO client VALUES(400, 'Dunmore Highschool', 2);

INSERT INTO client VALUES(401, 'Lackawana Country', 2);

INSERT INTO client VALUES(402, 'FedEx', 3);

INSERT INTO client VALUES(403, 'John Daly Law, LLC', 3);

INSERT INTO client VALUES(404, 'Scranton Whitepages', 2);
```

```
INSERT INTO client VALUES(405, 'Times Newspaper', 3);
INSERT INTO client VALUES(406, 'FedEx', 2);

-- WORKS_WITH
INSERT INTO works_with VALUES(105, 400, 55000);
INSERT INTO works_with VALUES(102, 401, 267000);
INSERT INTO works_with VALUES(108, 402, 22500);
INSERT INTO works_with VALUES(107, 403, 5000);
INSERT INTO works_with VALUES(108, 403, 12000);
INSERT INTO works_with VALUES(105, 404, 33000);
INSERT INTO works_with VALUES(107, 405, 26000);
INSERT INTO works_with VALUES(102, 406, 15000);
INSERT INTO works_with VALUES(105, 406, 130000);
```



Queries and Reports

Various queries are implemented to retrieve employee information, calculate total sales, count employees in each branch, and establish client-employee relationships.

1. **Retrieve the names of all employees.**

SELECT first_name, last_name FROM employee;

2. **Get the list of all branches.**

SELECT branch_name FROM branch;

3. **Display the names of clients.**

SELECT client_name FROM client;

4. **List all employees who have a supervisor.**

SELECT first_name, last_name FROM employee WHERE super_id IS NOT NULL;

5. **Find the total number of clients.**

SELECT COUNT(*) AS total_clients FROM client;

6. **Retrieve the branch name where employee with emp_id 107 works.**

SELECT branch_name FROM branch

WHERE branch_id = (SELECT branch_id FROM employee

WHERE emp_id = 107);

7. **Get the names of employees who were born in 1970.**

SELECT first_name, last_name FROM employee

WHERE YEAR(birth_day) = 1970;

8. **List all clients along with the branch names they are associated with.**

```
SELECT client.client_name, branch.branch_name FROM client
JOIN branch ON client.branch_id = branch.branch_id;
```

9. **Calculate the average salary of all employees.**

```
SELECT AVG(salary) AS average_salary FROM employee;
```

10. **Find the employee with the highest salary.**

```
SELECT emp_id, first_name, last_name, salary FROM employee ORDER BY salary DESC LIMIT 1;
```

11. **Count the number of employees in each branch.**

```
SELECT branch_id, COUNT(*) AS employee_count FROM employee GROUP BY branch_id;
```

12. **Retrieve the names of employees who work with the client "FedEx."**

```
SELECT first_name, last_name FROM employee
JOIN works_with ON employee.emp_id = works_with.emp_id
JOIN client ON works_with.client_id = client.client_id
WHERE client.client_name = 'FedEx';
```

13. **Display the branch names and the total number of employees in each branch.**

```
SELECT branch_name, COUNT(*) AS total_employees FROM branch
JOIN employee ON branch.branch_id = employee.branch_id
GROUP BY branch_name;
```

14. **Retrieve the names of employees who do not have a supervisor.**

```
SELECT first_name, last_name FROM employee
WHERE super_id IS NULL;
```

15. **Identify the employee with the highest total sales.**

```
SELECT emp_id, first_name, last_name, SUM(total_sales) AS total_sales FROM employee

LEFT JOIN works_with ON employee.emp_id = works_with.emp_id GROUP BY emp_id, first_name, last_name

ORDER BY total_sales DESC LIMIT 1;
```

16. Find the branch that has the highest average salary among its employees.

```
SELECT branch_id, AVG(salary) AS avg_salary FROM employee GROUP BY branch_id
ORDER BY avg_salary DESC LIMIT 1;
```

17. Retrieve the names of employees who work with more than one client.

```
SELECT first_name, last_name FROM employee

JOIN works_with ON employee.emp_id = works_with.emp_id GROUP BY employee.emp_id
HAVING COUNT(works_with.client_id) > 1;
```

18. List the names of employees who were born before their supervisors.

```
SELECT e1.first_name, e1.last_name FROM employee e1

JOIN employee e2 ON e1.super_id = e2.emp_id

WHERE e1.birth_day < e2.birth_day;
```

19. Calculate the total sales for each client.

```
SELECT client_id, client_name, SUM(total_sales) AS total_client_sales FROM works_with

JOIN client ON works_with.client_id = client.client_id

GROUP BY client_id, client_name;
```

20. Identify the branch with the highest total sales.

```
SELECT employee.emp_id, employee.first_name,
employee.last_name, SUM(works_with.total_sales) AS total_sales
FROM employee

LEFT JOIN works_with ON employee.emp_id = works_with.emp_id

GROUP BY employee.emp_id, employee.first_name,
employee.last_name

ORDER BY total_sales DESC LIMIT 1;
```

GitHub Link: [KarthicV358/MySQL-Employee-Database-Tracking-Sales-and-Relationships (github.com)]

Conclusion

In conclusion, the MySQL Employee Database project represents a successful endeavour in creating an efficient and comprehensive system for managing employee information, tracking sales, and analyzing organizational relationships. Leveraging the power of MySQL and SQL, the project has established a well-structured database with tables for Employee, Branch, Client, Works_With, and Branch_Supplier.

The database design, illustrated through the Entity-Relationship Diagram and detailed in the Database Schema section, forms the foundation for the project's success. It provides a clear representation of the interconnected entities and relationships, ensuring that the database captures and maintains data in a logical and organized manner.

The implementation phase, focused exclusively on MySQL and SQL, has resulted in a functional and responsive database system. The setup involved careful consideration of foreign key relationships, indexing key fields, and the strategic population of sample data. The execution of SQL queries, detailed in the Queries and Reports section, demonstrates the system's capability to retrieve employee information, calculate sales, and analyze branch performance.

By achieving its objectives of efficient employee data management, sales tracking, and relationship analysis, the MySQL Employee Database project offers a valuable tool for organizations seeking to optimize their internal processes. As technology evolves, this project serves as a testament to the enduring relevance and effectiveness of MySQL and SQL in developing robust database solutions. Future enhancements and optimizations can further refine and expand the project's capabilities, ensuring its continued utility in dynamic business environments.