# EMPLOYEE DATABASE MANAGEMENT SYSTEM

## FINAL REPORT

Abhilash Kosaraju | DBMS sec-06| 12/13/2017
|NEUID:001205393|

# Problem Statement:

Employee Database is the most widely deployed database seen in every organization. The employee database helps the organization to manage the employees and assists HR with workforce management. The employee database plays a critical role in analyzing the employee data and helps the organization to make critical HR decisions. I wish to design the replica of the employee database which holds employee details and employee activities with the help of various techniques discussed in the classroom. The main objective of this project is to implement an employee information system in an organization to return accurate information about employee activities by executing SQL queries in MySQL database

## PURPOSE:

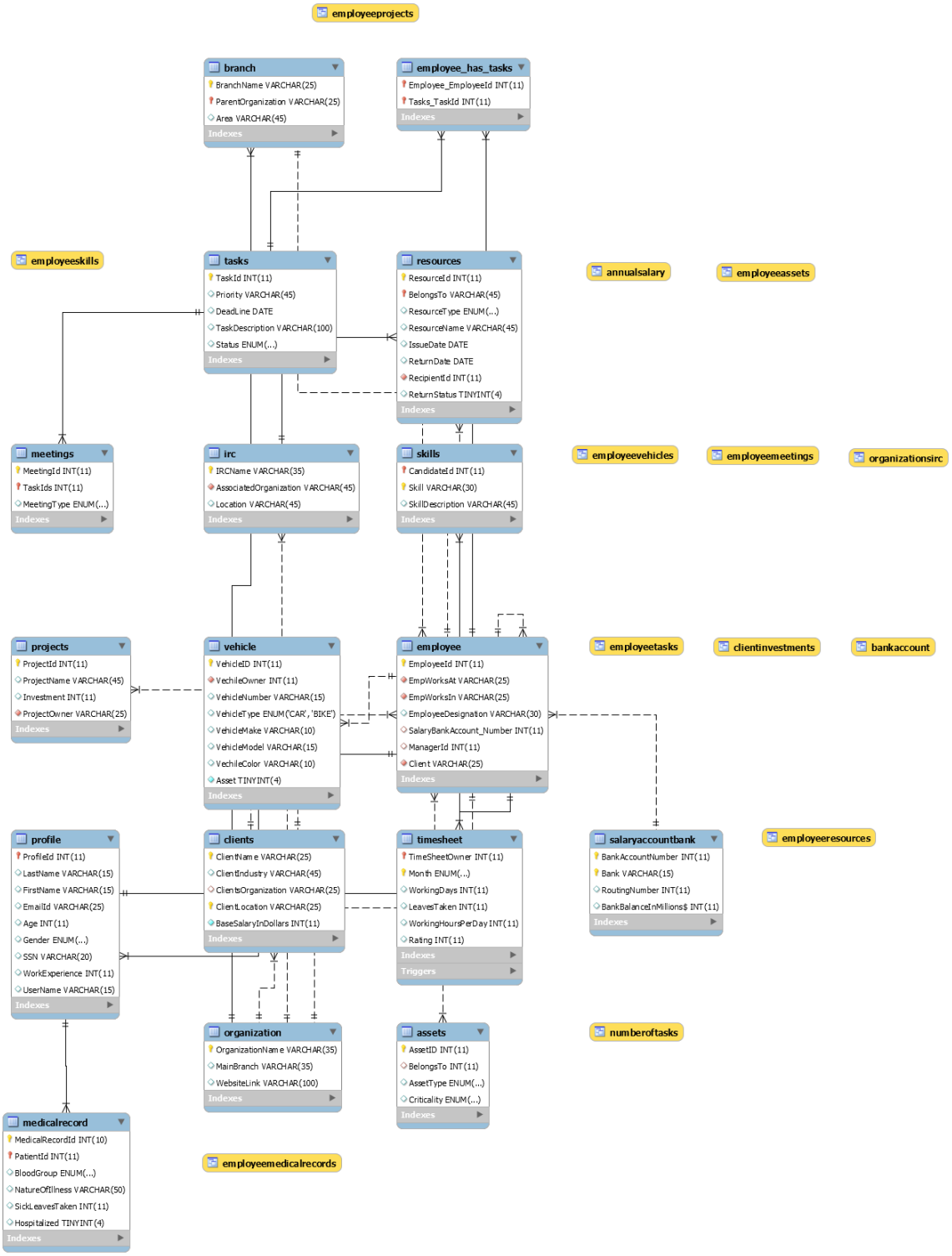
The proposed database design imitates the employee database as in maintaining employee profile, employee vehicle details, employee bank account details, employee medical records, employee skill sets and also manages the resources of organization's Information Resource Center(IRC).

## TECHNIQUES:

Stored Procedures are employed to perform data retrieval operations from the database. Views are created to present the related data. Triggers are employed to perform some calculations. Joins were mostly used to concatenate relevant details in subqueries. Functions are used to return aggregate values for analysis.

# ProJect demo:

EER Diagram

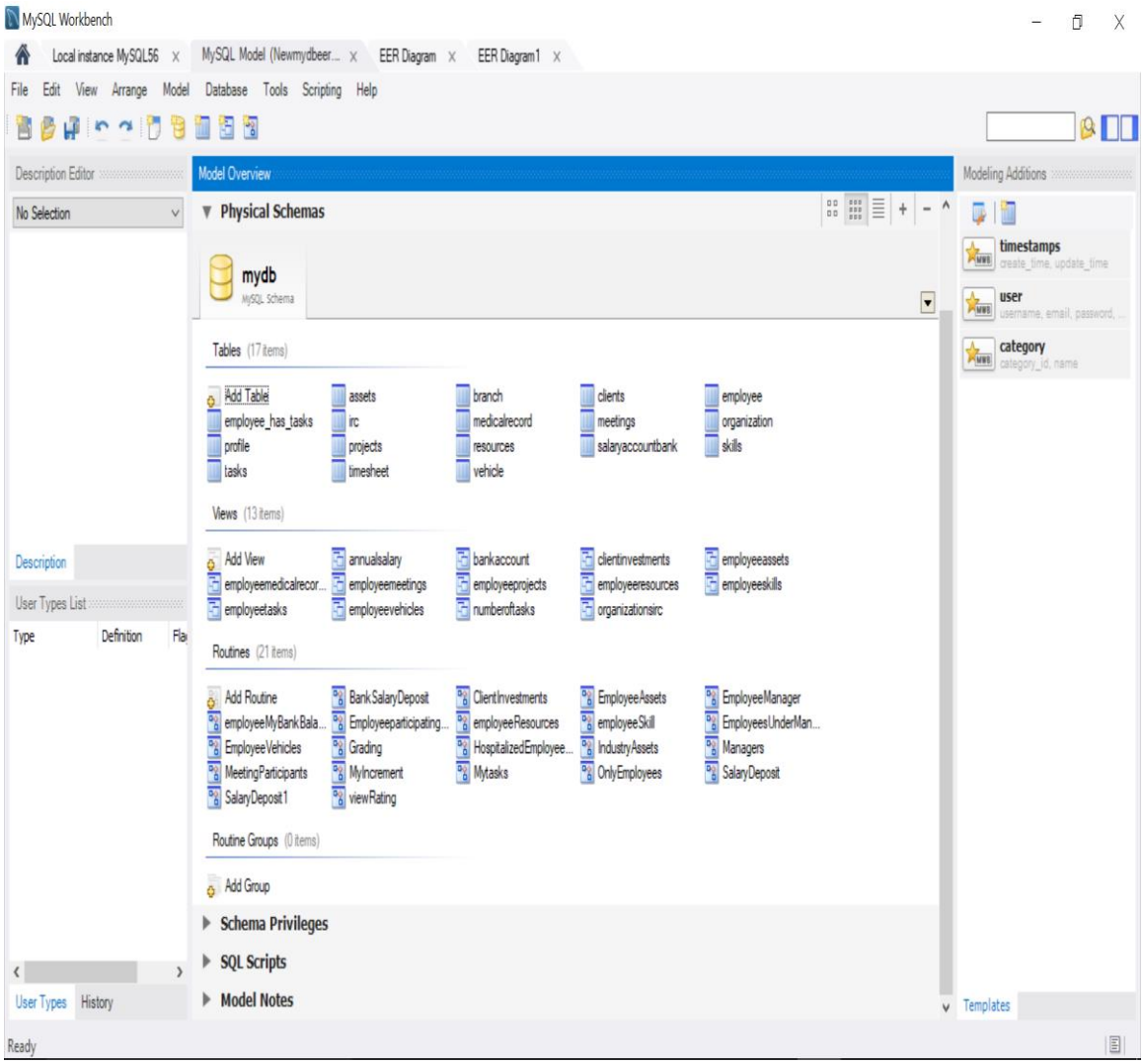


**The Physical Schema**

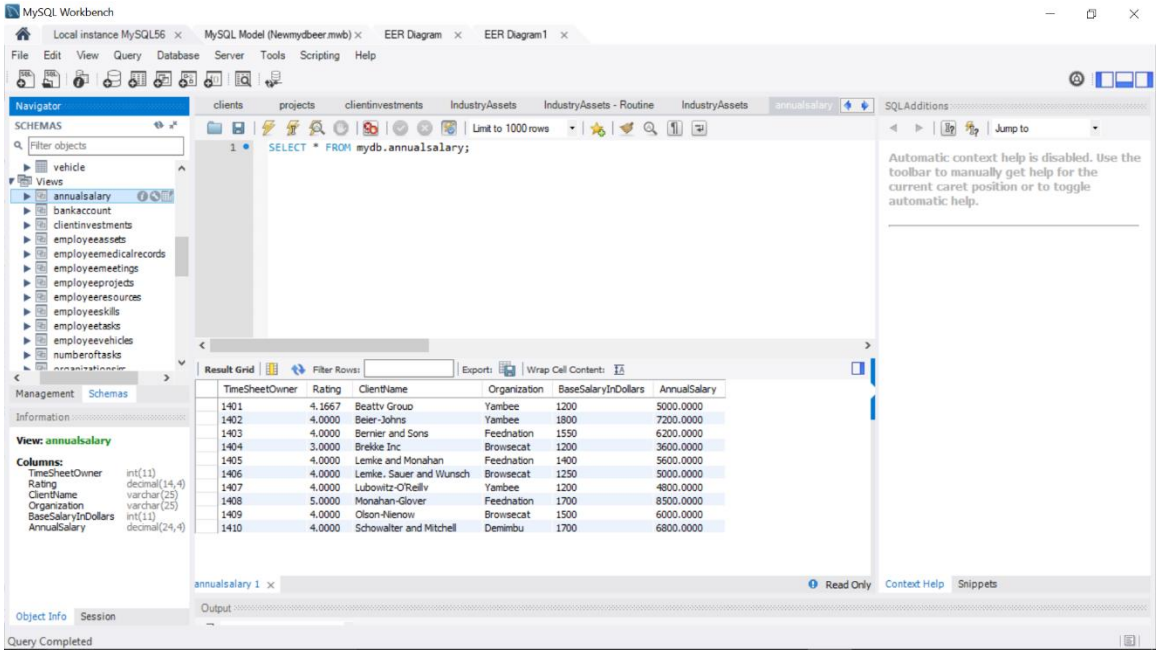
A total of 17 tables are defined and utilized in the database design.

13 views have been created to display the relevant data.

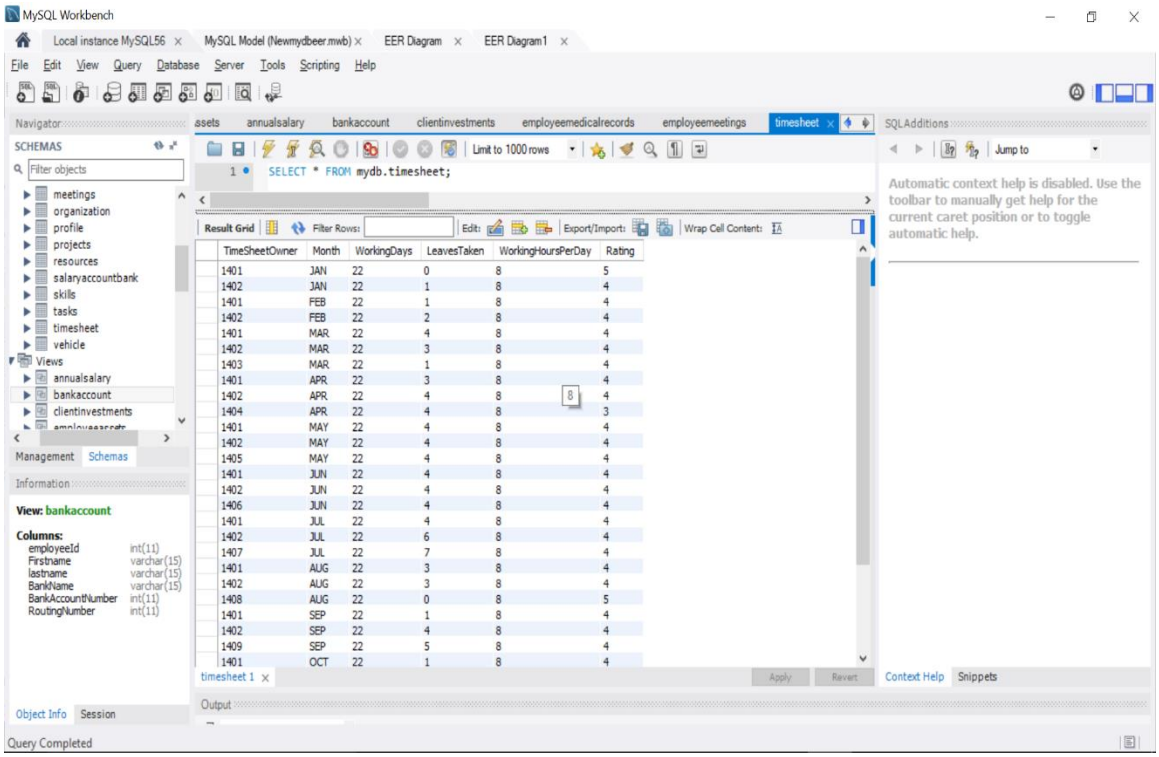
And 21 routines have been employed to retrieve the data and also to perform calculations for mathematical presentation of data.



## Employee Data




AnnualSalary will be calculated from the base salary defined in the client and the rating for the employee calculated in the Timesheet.



Trigger For calculating the Rating of the Employee based on the number of days an employee worked and leaves taken.

**CREATE DEFINER=`root`@`localhost` TRIGGER `mydb`.`timesheet_BEFORE_UPDATE` BEFORE UPDATE ON `timesheet` FOR EACH ROW**

**BEGIN**

**If ((new.WorkingHoursPerDay - new.LeavesTaken)*(new.workingdays) = 176) then set new.Rating = 5;**

**elseif (160< (new.WorkingHoursPerDay- new.LeavesTaken)*(new.workingdays) < 176) then set new.Rating =4;**

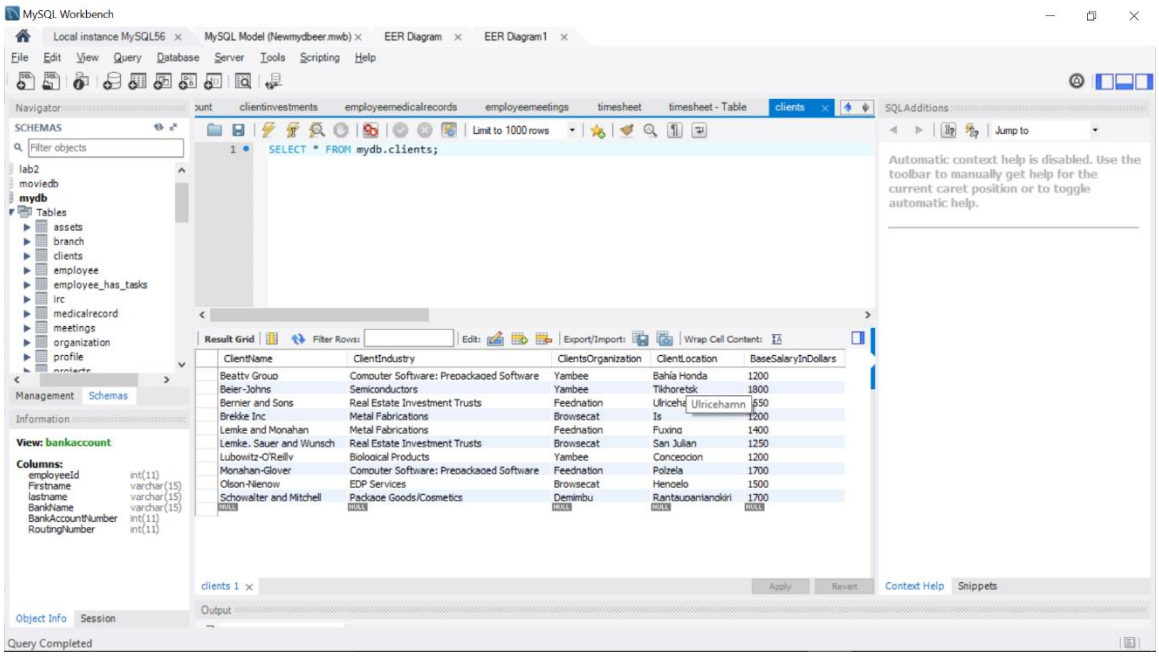
**elseif (144 < (new.WorkingHoursPerDay- new.LeavesTaken)*(new.workingdays) < 160) then set new.Rating=3;**

**elseif ( 128 <(new.WorkingHoursPerDay- new.LeavesTaken)*(new.workingdays) < 144) then set new.Rating=2;**

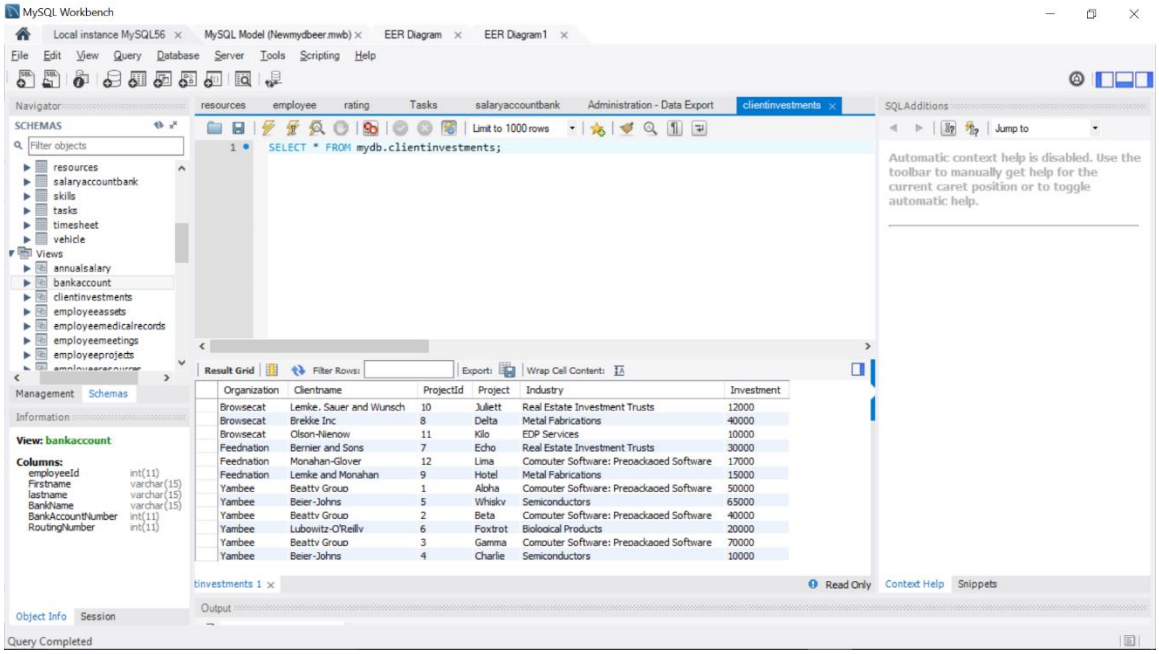
**else**

**set new. Rating=1;**

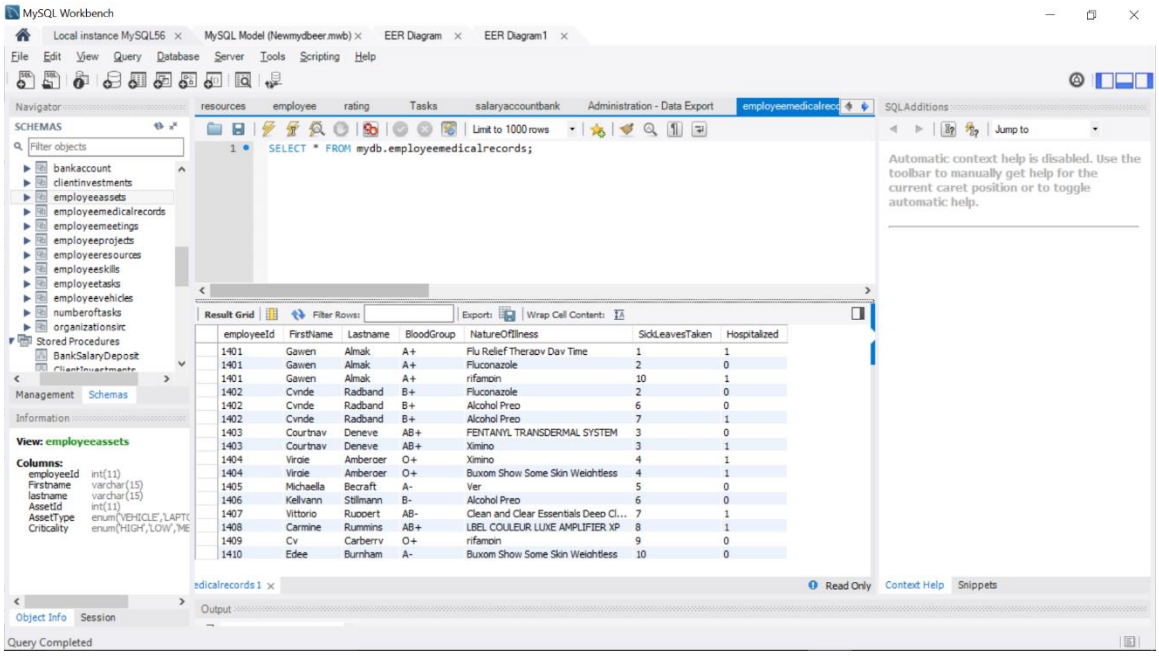
**end if;**



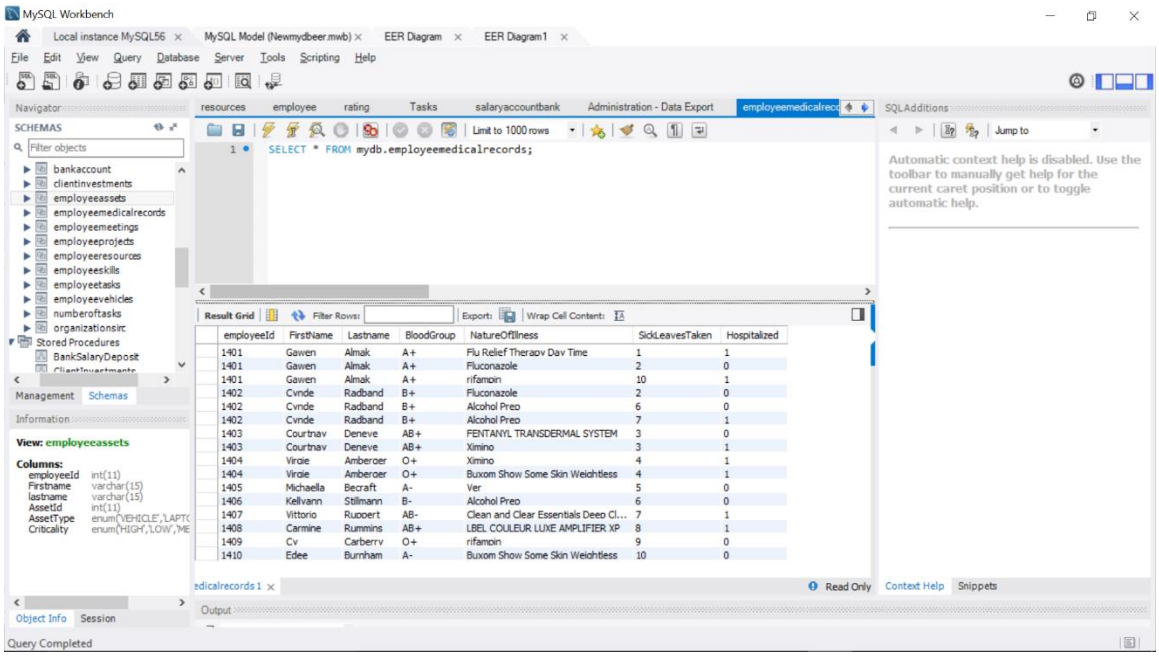
## Client Investments in the Organization



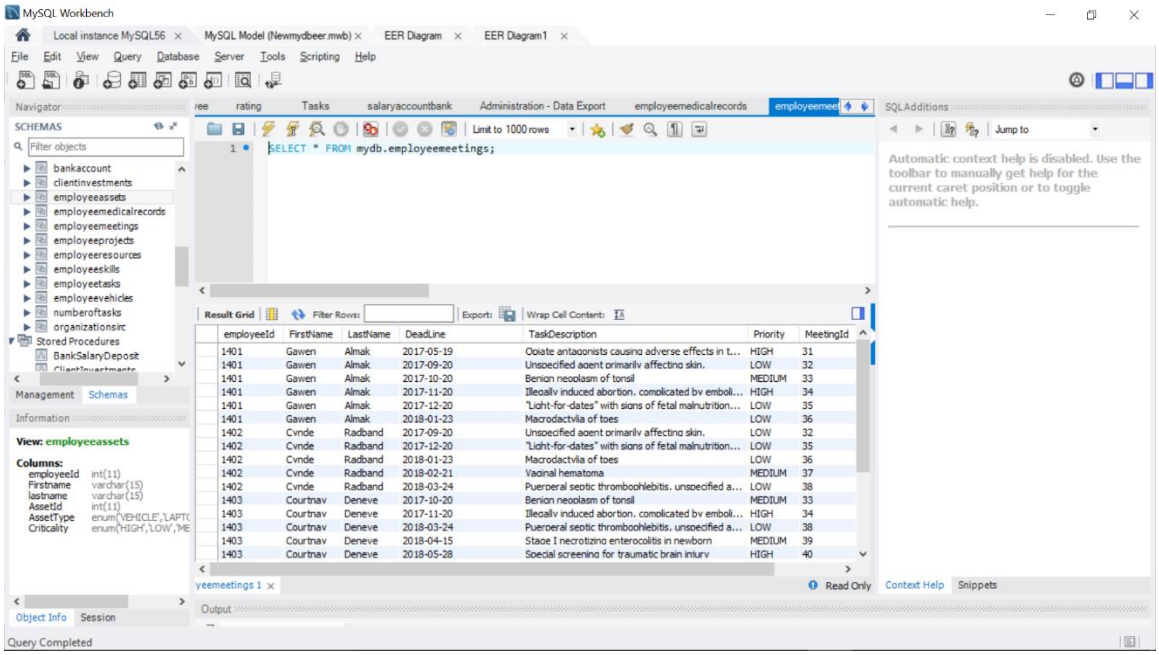

## Employee Medical Records



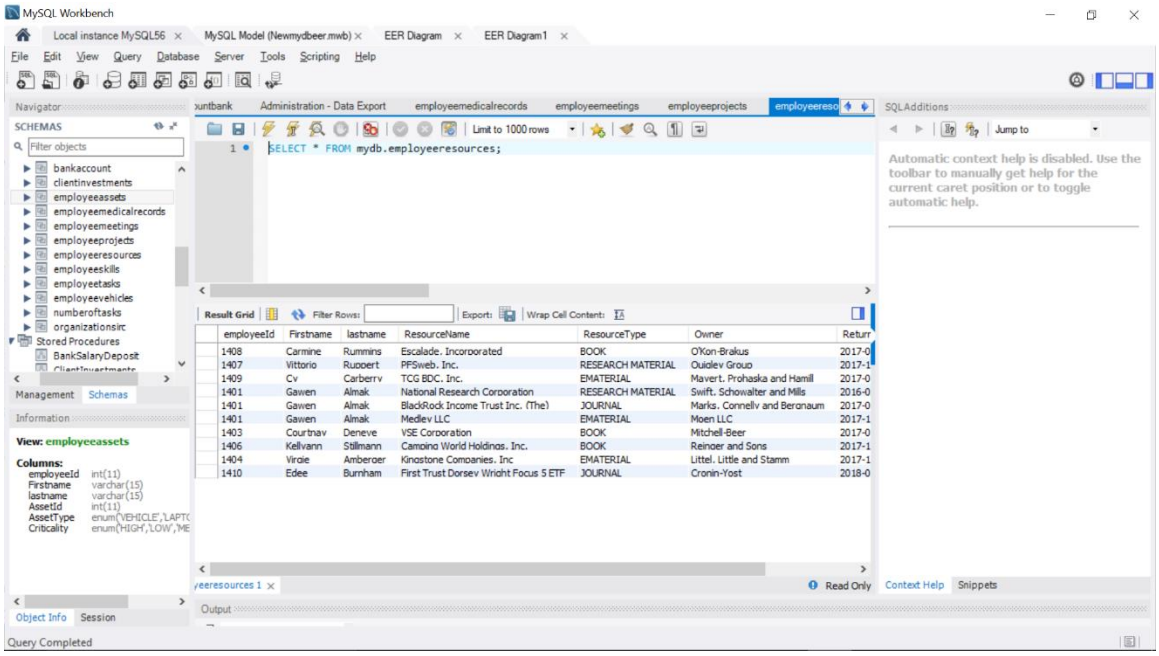
## Employee Meetings



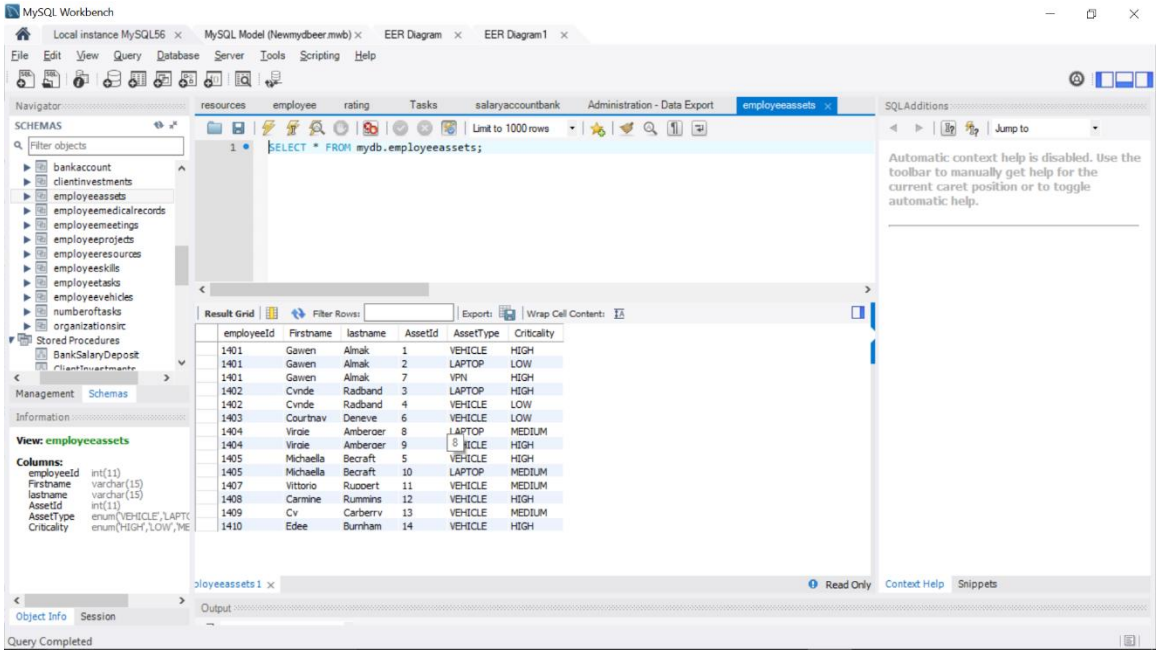

## Employee Projects



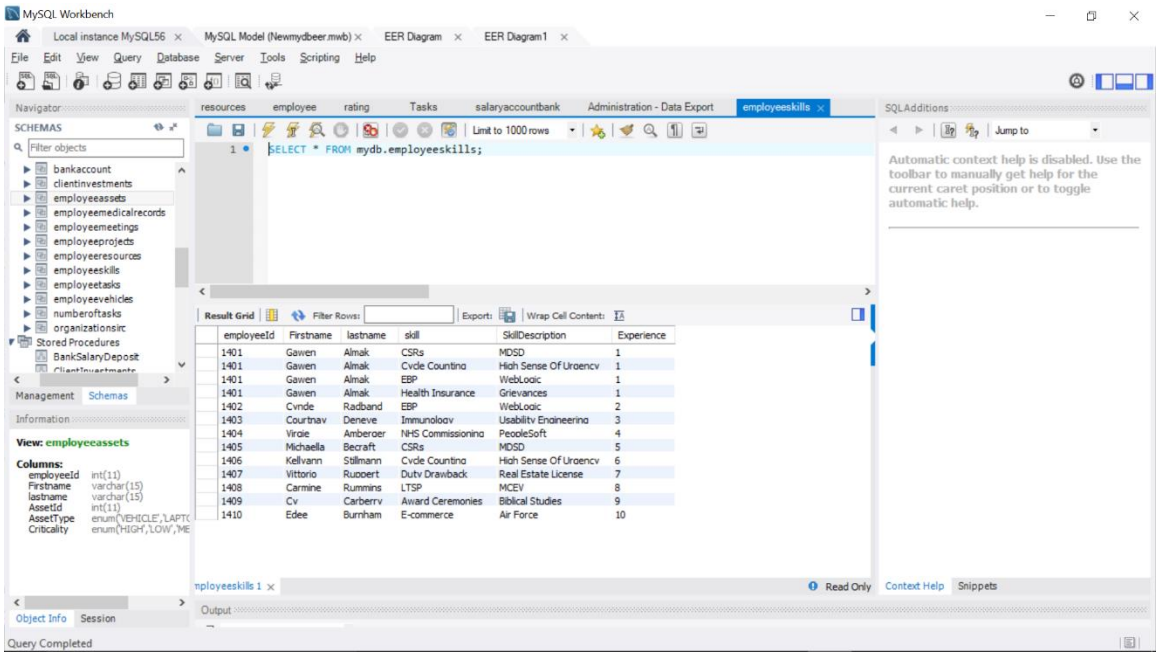
## Employee's Resources from Information Resource Center associated with Organization



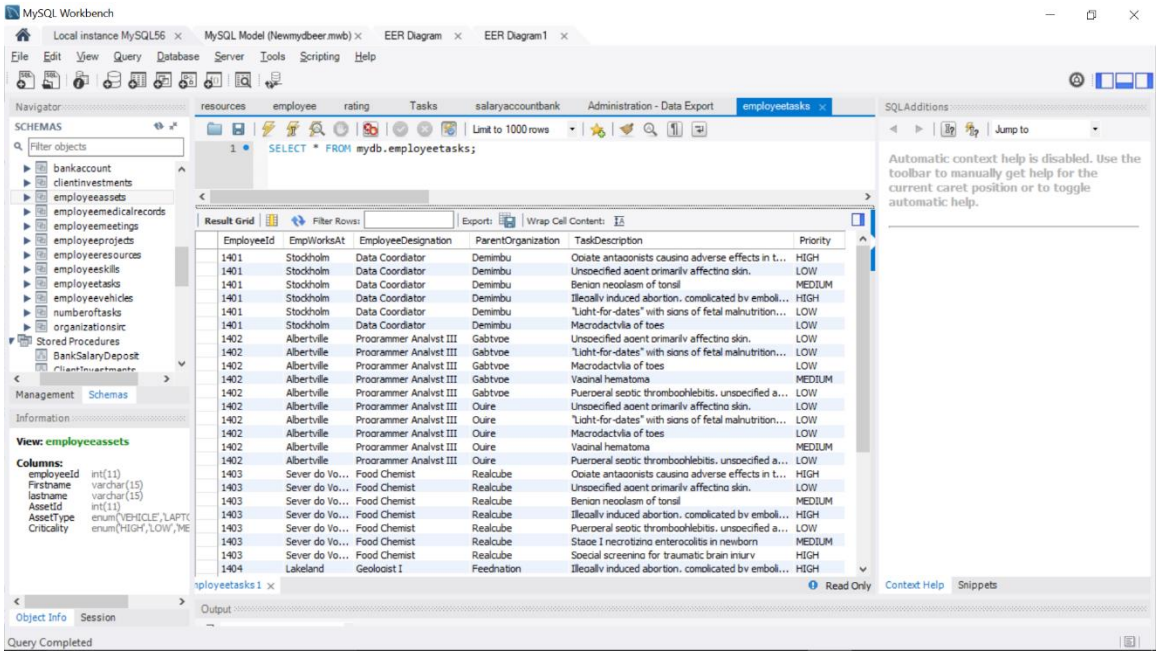

## Employee Assets:



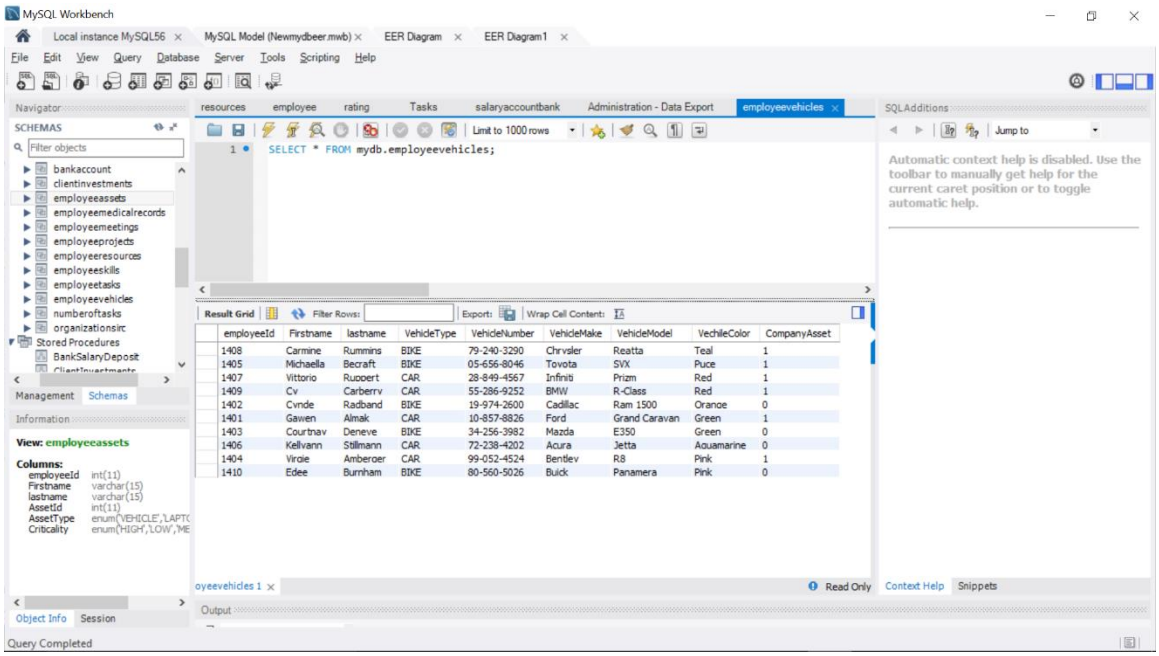
## Employee Skills and WorkExperience



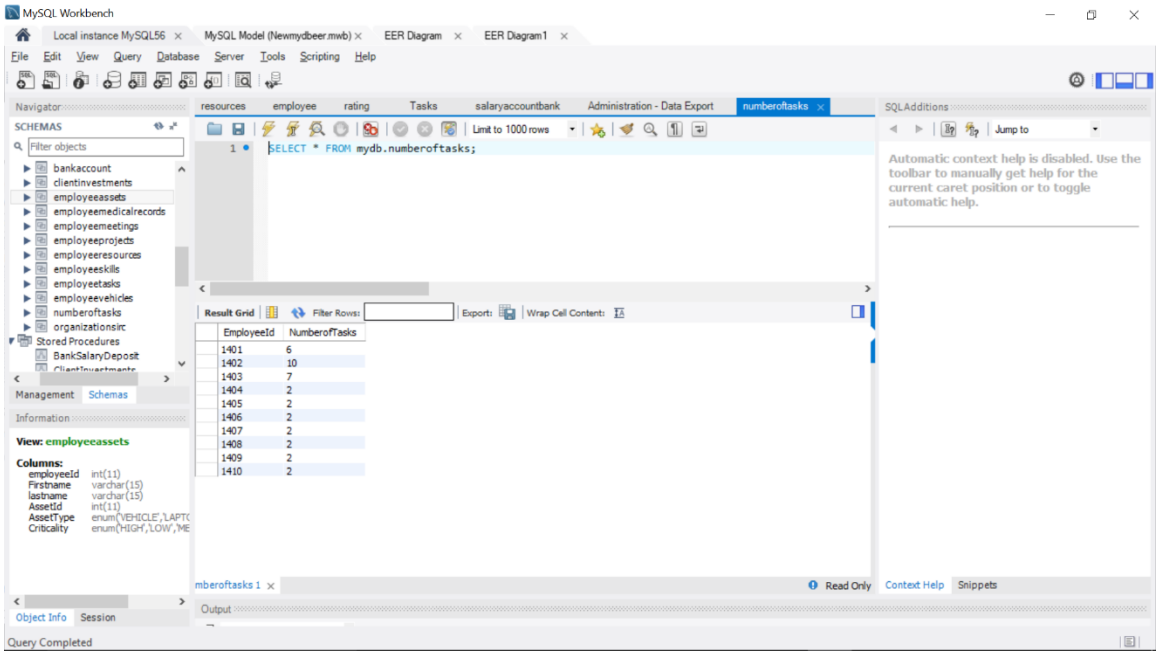

## Employee's Tasks



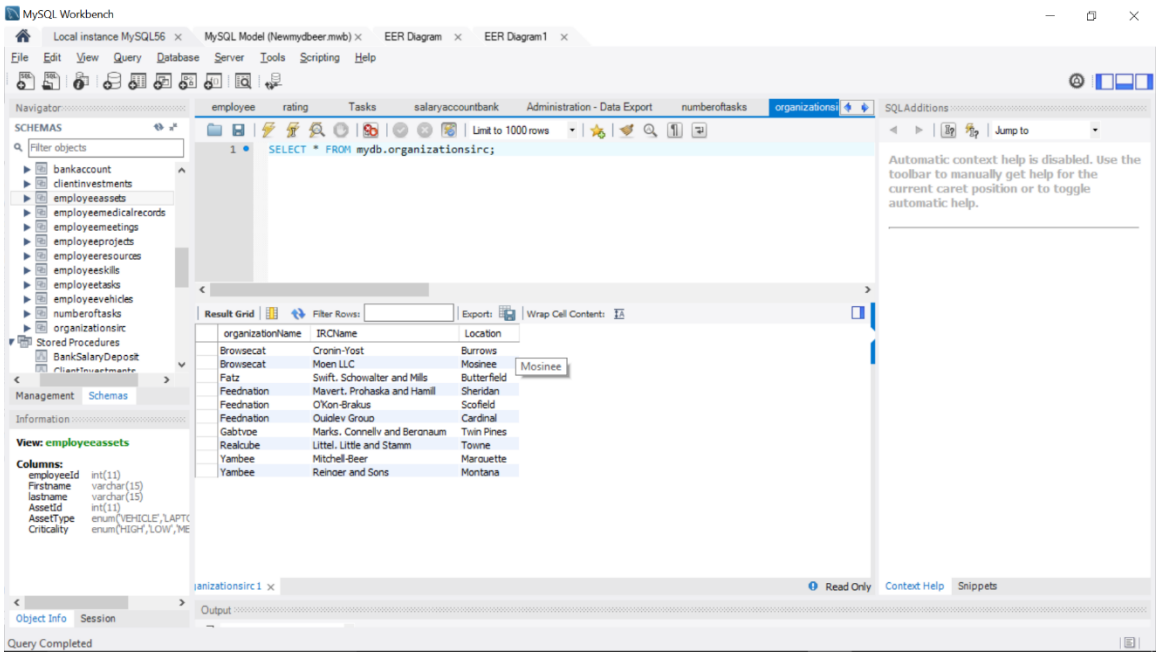
## Employee Vehicles




## Number of Employee Tasks grouped by EmployeeId



**Information Resource Centers associated with Organizations**




**STORED PROCEDURES**

**BankSalaryDeposit**

Deposits the incremented salary in the employees SalaryBankAccount

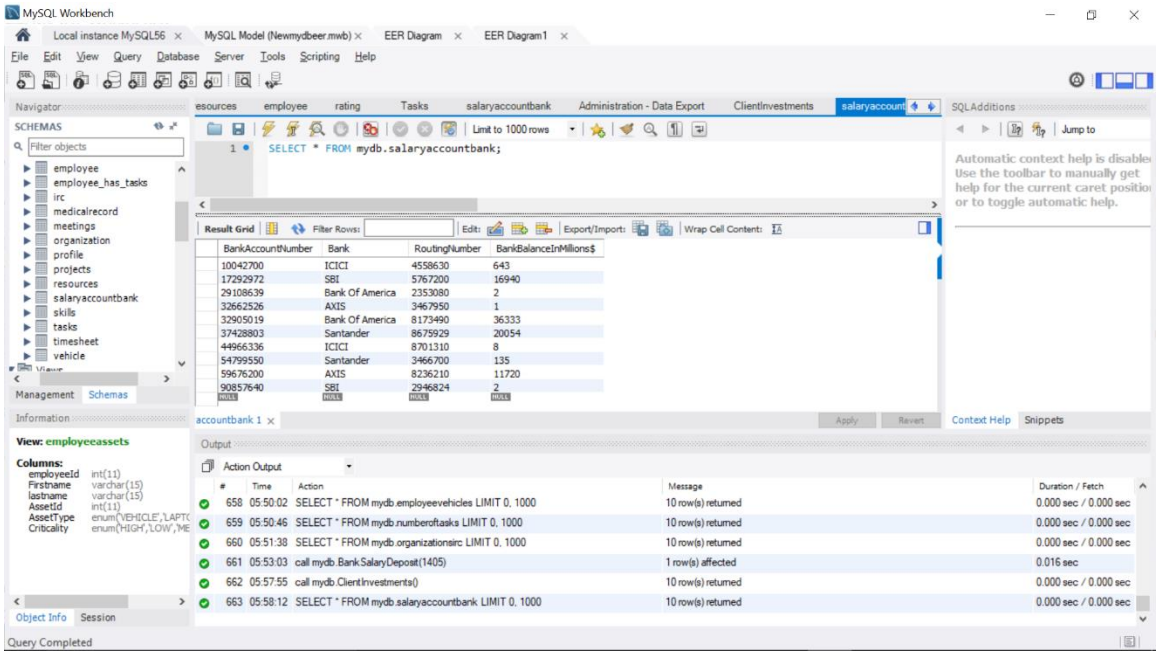
**CREATE DEFINER=`root`@`localhost` PROCEDURE `BankSalaryDeposit`(IN id int)**

**BEGIN**

**update salaryaccountbank set BankBalanceInMillions$ = BankBalanceInMillions$ + SalaryDeposit1 (id ) where**

**BankAccountNumber=(select SalaryBankAccount_Number from employee where EmployeeId =id);**

**END**




**Client investments in Organization**

**CREATE DEFINER=`root`@`localhost` PROCEDURE `ClientInvestments`()**

**BEGIN**

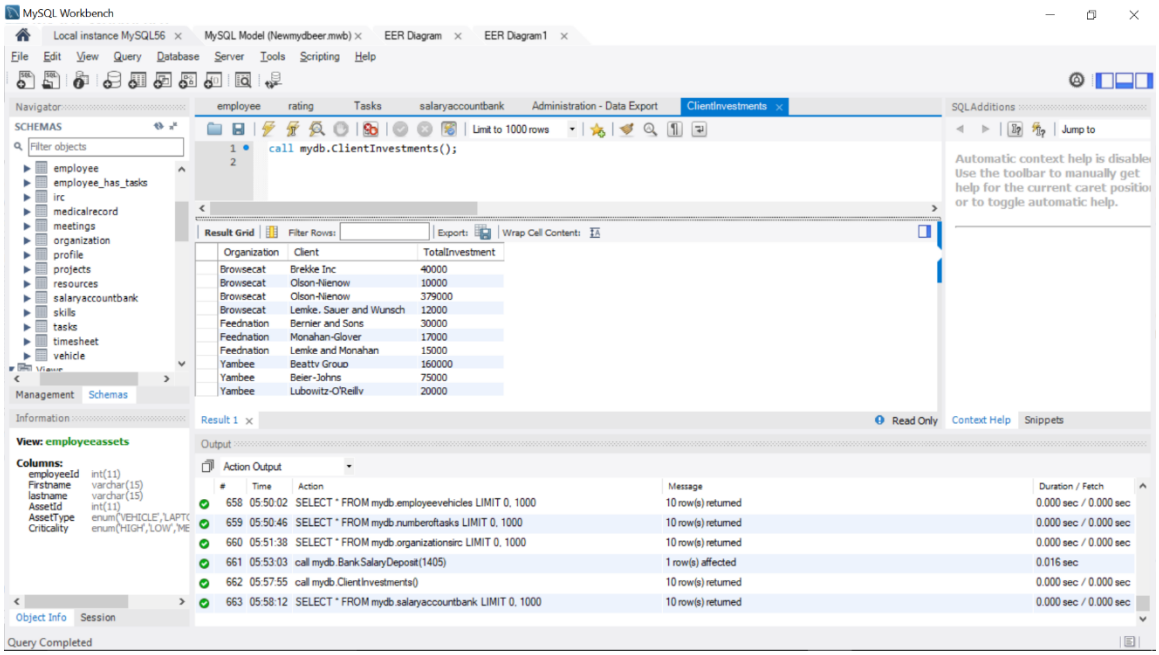
**select IFNULL(Organization, 'Org') AS Organization,**

**IFNULL(Clientname, 'Total') As Client, sum(investment) As TotalInvestment**

**from clientinvestments**

**Group by ClientName with rollup;**

**END**




**EmployeeManager**

Employee's Manager details will be displayed

**CREATE DEFINER=`root`@`localhost` PROCEDURE `EmployeeManager`(IN id int)**

**BEGIN**

**select * from employee**

**inner join**

**profile**

**on employee.EmployeeId = ProfileId**

**where EmployeeId in (SELECT ManagerId**

**FROM employee**

**where EmployeeId= id);**

**END**

**EmployeeAssets**

Retrieves all the assets of the employee.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EmployeeAssets`(IN id int)
BEGIN
select  employeeId,Firstname,lastname,AssetId,AssetType,Criticality
from employee
inner join profile
on employee.EmployeeId = profile.ProfileId
inner join assets
on employee.employeeId = assets.BelongsTo
where BelongsTo = id
order by EmployeeId;
END
```

**EmployeeMyBankBalance**

Gives the employee, the total amount in his bank account of the employee

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `employeeMyBankBalance`(IN id
int)
BEGIN
select employeeId,FirstName,lastname,b.BankAccountNumber as
AccountNumber,Bank,BankBalanceInMillions$ as mybalance
from bankaccount b
inner join
salaryaccountbank
on b.BankAccountNumber = salaryaccountbank.BankAccountNumber
where employeeId =id;
END
```

**EmployeeparticipatingProjects**

Presents all the working projects of the employees

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`EmployeeparticipatingProjects`(IN id int)

BEGIN

select * from EmployeeProjects

where employeeId =1401;

END
```

**employeeResources**

Displays all the resources with employee.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `employeeResources`(IN id int)

BEGIN

select  employeeId,Firstname,lastname,ResourceName,ResourceType,BelongsTo As
Owner,ReturnDate,ReturnStatus

from employee

inner join profile

on employee.EmployeeId = profile.ProfileId

inner join resources

on employee.employeeId = resources.recipientId

where RecipientId = id;

END
```

**Employee Skills**

Skills of the concerned Employee

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `employeeSkill`(IN id int)
BEGIN
select  *
from employeeskills
where
employeeId=id;
END
```

**EmployeesUnderManager**

Displays all the employees under the employee(Manager)

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EmployeesUnderManager`(IN id int)
BEGIN
SELECT *
FROM employee inner join
profile
on employee.EmployeeId = profile.ProfileId
where ManagerId = 1401;
END
```

**EmployeeVehicles**

All the vehicles of the employee.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `EmployeeVehicles`(IN id int)

BEGIN

select
employeeId,Firstname,lastname,VehicleType,VehicleNumber,VehicleMake,VehicleMo
del,VechileColor, Asset as CompanyAsset

from employee

inner join profile

on employee.EmployeeId = profile.ProfileId

inner join vehicle

on employee.employeeId = vehicle.VechileOwner

where VechileOwner = id;

END
```

**Grading**

Assigns the grades to the employee based on the number of task the employee is working with.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Grading`(IN id int, OUT Rating
varchar(5))

BEGIN

case

when (select numberoftasks.numberofTasks from numberoftasks where
employeeId=id ) =10

then set rating ='A';

when (select numberoftasks.numberofTasks from numberoftasks where
employeeId=id ) >=5 AND (select numberoftasks.numberofTasks from numberoftasks
where employeeId=id) <10

then set rating ='B';
```

```
else set rating ='C';

end case;

END
```

**HospitalizedEmployeeMedicalRecords**

Gives the hospitalized medical records of the employee

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`HospitalizedEmployeeMedicalRecords`(IN id int)

BEGIN

select *

from EmployeeMedicalRecords

where Hospitalized = 1

AND employeeId=id

order by EmployeeId;


END
```

**IndustryAssets**

Displays the stakes of  Organization in different industries.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `IndustryAssets`()
BEGIN

select IFNULL(Organization, 'Org') AS Organization,
IFNULL(Industry,'Total') As ClientIndustry,
sum(investment) As TotalInvestment
from clientinvestments
Group by  Industry,Organization with rollup;
END
```

**MeetingParticipants**

Displays the participants of the meeting when a meeting input is given

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `MeetingParticipants`(IN id int)
BEGIN
select employeeId,FirstName,Lastname,Priority as
TaskPriority,MeetingType,DeadLine as TaskDeadline
from employeemeetings
where meetingId =id;
END
```

**MyIncrement**

Gives the amount of increment the employee has got, based on the grade he has earned.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `MyIncrement`(IN id int, OUT SalaryWithIncrement int)

BEGIN

case

when (select numberoftasks from numberoftasks where employeeId=id ) =10

then set SalaryWithIncrement = ((select annualsalary.AnnualSalary  from annualsalary where annualsalary.TimeSheetOwner=id)*5);


when (select numberoftasks.numberoftasks from numberoftasks where employeeId=id ) >=5 AND (select  numberoftasks.numberoftasks.numberoftasks from numberoftasks where employeeId=id) <10

then set SalaryWithIncrement =((select annualsalary.AnnualSalary  from annualsalary where annualsalary.TimeSheetOwner=id)*4);


else set SalaryWithIncrement =((select annualsalary.AnnualSalary  from annualsalary where annualsalary.TimeSheetOwner=id)*3);


end case;

END
```

**Mytasks**

Displays his tasks to the employee

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Mytasks`(IN id int)
BEGIN
select * from employeeTasks
where
employeeid =id;
END
```

**OnlyEmployees**

Displays only the employees in the organization and not the managers

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `OnlyEmployees`()
BEGIN
SELECT *
FROM employee inner join
profile
on employee.EmployeeId = profile.ProfileId
where ManagerId IS NOT NULL;
END
```

**OnlyManagers**

Displays only the mangers

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Managers`()
BEGIN
SELECT *
FROM employee inner join
profile
on employee.EmployeeId = profile.ProfileId
where ManagerId IS NULL;
END
```

**ViewRating**

Average ratings for the employees would be shown in the descending order.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `viewRating`()
BEGIN
  SELECT TimesheetOwner as EmployeeId,rating,clientname,Organization FROM annualsalary
  group by organization,clientname
  order by rating desc;
  END
```

**FUNCTION**

**SalaryDeposit**

Returns the amount to deposited in the employee bank account. Used in the BankSalaryDeposit stored procedure.

**CREATE DEFINER=`root`@`localhost` FUNCTION `SalaryDeposit`(id int) RETURNS int(11)**

**BEGIN**

**DECLARE deposit int;**

**case**

**when (select numberoftasks from numberoftasks where employeeId=id ) =10**

**then set deposit = ((select annualsalary.AnnualSalary from annualsalary where annualsalary.TimeSheetOwner=id)*5);**

**when (select numberoftasks.numberoftasks from numberoftasks where employeeId=id ) >=5 AND (select numberoftasks.numberoftasks.numberoftasks from numberoftasks where employeeId=id) <10**

**then set deposit =((select annualsalary.AnnualSalary from annualsalary where annualsalary.TimeSheetOwner=id)*4);**

**else set deposit =((select annualsalary.AnnualSalary from annualsalary where annualsalary.TimeSheetOwner=id)*3);**

**end case;**

**return deposit;**

**end**

**TO CONCLUDE:**

In the above database design, I have tried to address all the significant functionalities of employee database to present the completed system.