DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

|  |  |
| --- | --- |
| **PES2201800331** | **Chethan M** |

|  |
| --- |
| Payroll (salary) management is implement here using Structured Query Language (SQL). The data model is created such that it contains 6 relations namely HR, accountant, salary, transaction, funds and employee. This database design not only stores and explains about the salary but also the information like who assigns it, who has the rights to add/delete employee etc.    Tables are created accurately such that all constrains like referential integrity constrains, check constrains, entity integrity are taken at most care. Primary and foreign keys are properly defined so as to have smooth transactions and proper normalization of tables.  Triggers are used here have a dynamic display features so that user doesn’t have to type queries to display the tables. Once he updates a table, the table is displayed instantly so that he can think of further modifications and conclusions.  On the conclusion note, this database mimics the original large databases used in companies to store data and assign salaries to their employees in a simpler structural way. |

**INTRODUCTION**

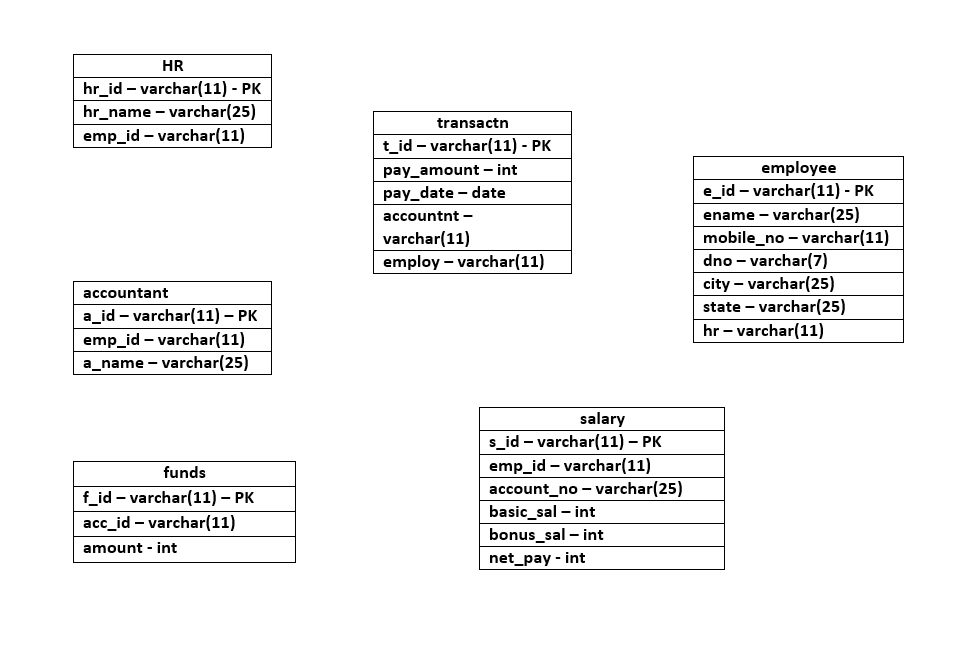
The main objective of this project is to demonstrate the way how payroll (salary) management is used in companies to assign salaries to their employees and also to mimic the process of allocation of funds to the employees.

There are five entities used here:

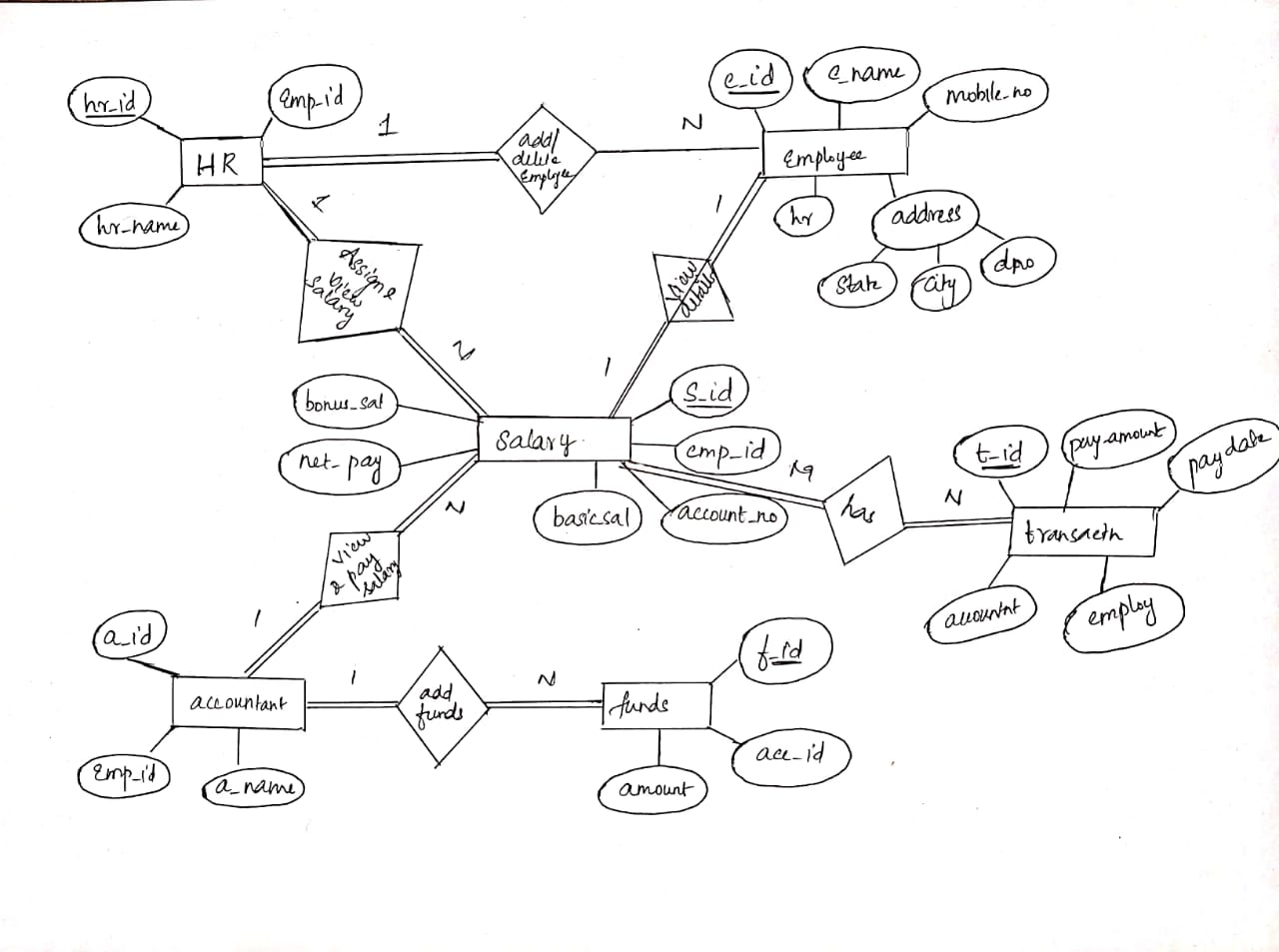
|  |  |
| --- | --- |
| **HR** | add/delete employee  assign/view salaries |
| **ACCOUNTANT** | view/pay salaries  add funds |
| **EMPLOYEE** | view their salaries |
| **SALARY** | contains details of all salaries given  also contains derived column - **transaction** |
| **FUNDS** | contains details about total funds allocated to assign salaries |

**DATA MODEL**

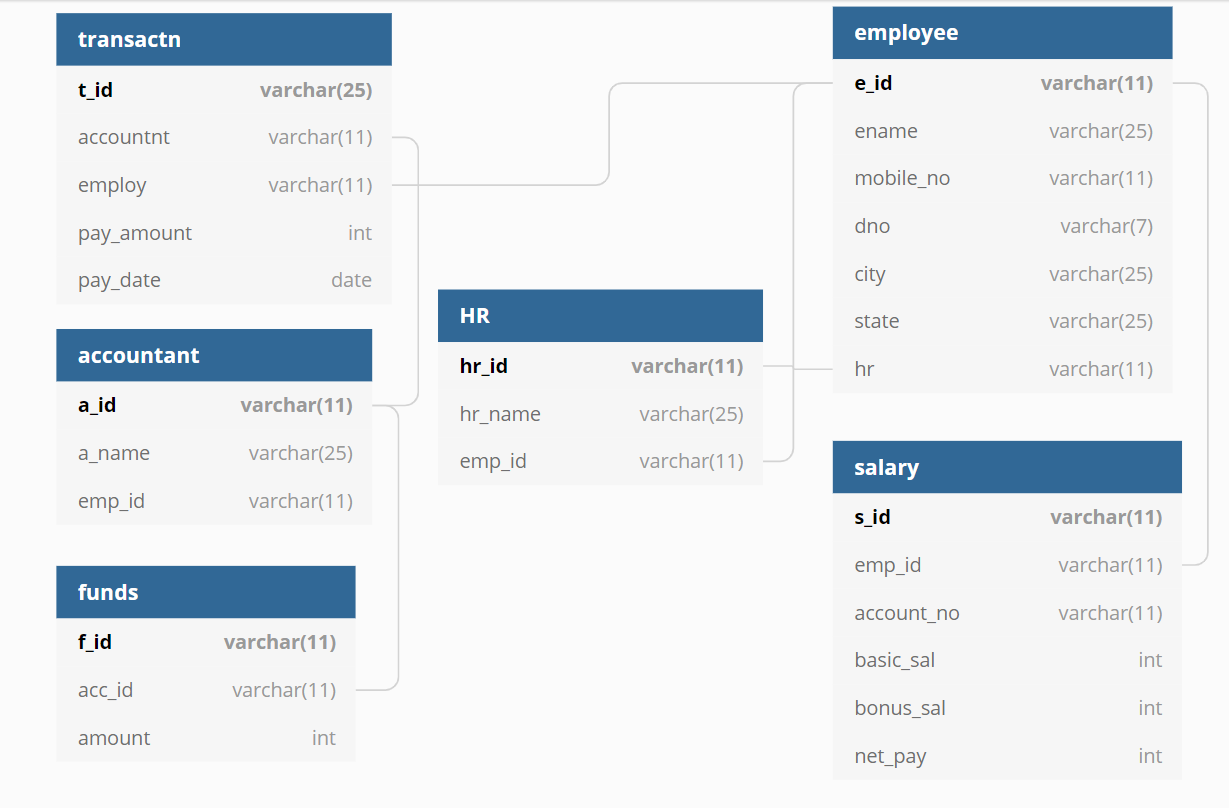
**1. VARIABLE DESCRIPTION**

****

**2. ER DIAGRAM**

****

**3. SCHEMA DESIGN**

****

**FUNCTIONAL DEPENDENCIES**

**1. HR**

1. hr\_id -> emp\_id

2. hr\_id -> hr\_name

**SUPER KEY**: (hr\_id, emp\_id), (hr\_id, hr\_name) etc.

**CANDIDATE KEY**: hr\_id

**NORMAL FORM: BOYCE-CODD NORMAL FORM**

**2. transactn**

1. t\_id -> accountnt

2. t\_id -> pay\_amount

3. t\_id -> pay\_date

4. t\_id -> employ

**SUPER KEYS**: (t\_id, employ), (t\_id, accountnt) etc.

**CANDIDATE KEY**: t\_id

**NORMAL FORM: 3RD NORMAL FORM**

**3. accountant**

1. a\_id -> emp\_id

2. a\_id -> a\_name

3. emp\_id -> a\_name

**SUPER KEYS**: (a\_id, emp\_id), (a\_id, a\_name) etc.

**CANDIDATE KEY**: a\_id

**NORMAL FORM: BOYCE-CODD NORMAL FORM**

**4. employee**

1. e\_id -> ename

2. e\_id -> mobile\_no

3. e\_id -> hr

**SUPER KEYS**: (e\_id, ename), (e\_id, hr) etc.

**CANDIDATE KEY**: e\_id

**NORMAL FORM: 3RD NORMAL FORM**

**5. funds**

1. f\_id -> acc\_id

2. f\_id -> amount

**SUPER KEYS**: (f\_id, acc\_id), (f\_id, amount) etc.

**CANDIDATE KEY**: f\_id

**NORMAL FORM: BOYCE-CODD NORMAL FORM**

**6. salary**

1. s\_id -> emp\_id

2. s\_id -> net\_pay

3. emp\_id -> account\_no

**SUPER KEYS**: (s\_id, emp\_id), (s\_id, account\_no) etc.

**CANDIDATE KEY**: s\_id

**NORMAL FORM: 3RD NORMAL FORM**

**NORMALIZATION**

**1. First Normal form**

All the tables mentioned above are in first normal form since all the values are in atomic and columns have data of same data type.

Address attribute of employee table is a composite variable, hence it is broken down into its atomic columns. If not broken down, it violates the rules of first normal form.

Mobile\_no attribute of employee is not used as multi value attribute since it violates the atomic rule of first normal form. If it is used as a multi value attribute, then the table has to be broken down into two tables or the columns must be split into multiple columns to have atomic values in them. But here mobile\_no is used as an atomic attribute, i.e. it contains a single mobile number.

**2. Second Normal form**

Candidate keys of all the columns contain single attribute in it, hence all the relations of the database are in second normal form.

If any relation has combination of attributes as primary key, it might violate the rules of 2nd normal form.

**3. Third Normal form**

The salary relation doesn’t obey the rules of third normal form since it has a transitive dependency in it.

Hence the relation has to be split into two tables.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s\_id | emp\_id | account\_no | basic\_pay | bonus\_pay | net\_pay |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s\_id | emp\_id | basic\_pay | bonus\_pay | net\_pay |

|  |  |
| --- | --- |
| emp\_id | account\_no |

Similarly the accountant relation also has to be broken down into two tables since it has a transitive relation in it (violates the 3NF).

**4. Boyce – Codd Normal form**

All relations except salary and accountant tables have determinants of non-trivial Functional Dependencies as super keys. Hence they all follow BC Normal form.

If the changes suggested in 3NF are done for both accountant and salary relations then all the relations of the salary management database are in highest normal form possible (BCNF).

**LOSSLESS JOIN TEST**

For the lossless join test, “salary” relation is taken as an example.

salary (s\_id, emp\_id, account\_no, basic\_pay, bonus\_pay, net\_pay)

FD (s\_id) -> {emp\_id, account\_no, basic\_pay, bonus\_pay, net\_pay)

salary relation is divided into:

salary1 (s\_id, emp\_id, basic\_pay, bonus\_pay, net\_pay)

salary2 (emp\_id, account\_no)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | s\_id | emp\_id | account\_no | basic\_pay | bonus\_pay | net\_pay |
| salary1 | **α** | **α** | **β** | **α** | **α** | **α** |
| salary2 | **β** | **α** | **α** | **β** | **β** | **β** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | s\_id | emp\_id | account\_no | basic\_pay | bonus\_pay | net\_pay |
| salary1 | **α** | **α** | **α** | **α** | **α** | **α** |
| salary2 | **β** | **α** | **α** | **β** | **β** | **β** |

**Since all columns of the relation salary1 have α in them, the test for lossless join property is passed.**

**Hence, the relation is lossless and dependency preserving.**

**DATA DEFINITION LANGUAGE**

**Structured Query Language (SQL)** is used to implement the payroll management database.

**CODE**:

CREATE Database Salary\_Management

Create table employee

(

e\_id VARCHAR(11) primary key,

ename varchar(25) NOT NULL,

mobile\_no varchar(11) NOT NULL,

dno varchar(10) NOT NULL,

city varchar(20) NOT NULL,

St varchar(25) NOT NULL,

);

Create table accountant

(

a\_id varchar(11) primary key not null,

emp\_id varchar(11) not null,

a\_name varchar(25) not null,

constraint fk\_emp\_id

foreign key (emp\_id)

references employee (e\_id)

);

create table HR

(

hr\_id varchar(11) primary key not null,

hr\_name varchar(25) not null,

emp\_id varchar(11) not null,

constraint fk\_emp\_id1

foreign key (emp\_id)

references employee(e\_id)

);

create table transactn

(

t\_id varchar(11) primary key not null,

pay\_amt int not null ,

pay\_date date not null,

accountnt varchar(11) not null,

employ varchar(11) not null,

constraint ct2

foreign key (accountnt) references accountant (a\_id),

foreign key (employ) references employee (e\_id)

);

create table funds

(

f\_id varchar(11) primary key not null,

ac\_id varchar(11) not null,

amount int not null,

constraint ct3

foreign key (ac\_id) references accountant(a\_id)

);

create table salary

(

s\_id varchar(11) primary key not null,

emp\_id varchar(11) not null,

account\_no varchar(20) unique not null,

basic\_sal int not null,

bonus\_sal int not null,

net\_pay int not null

constraint new1

foreign key (emp\_id) references employee (e\_id)

)

insert into employee values('1','John','8643712498','4/3A','Bangalore','KA',);

insert into employee values('2','Ram','9432117898','3/4B','Bangalore','KA',);

insert into employee values('3','Raj','9765132139','77/C','Hyderabad','TS',);

insert into employee values('4','Abdul','8866211318','8/7C','Pune','MH','1');

insert into employee values('5','Khan','8778771432','4C/7','Mumbai','MH');

select \* from employee

insert into accountant values('1','3','Raj')

select \* from accountant

insert into HR values('1','Khan','5')

select \* from HR

insert into transactn values('1',10000,'2020-05-03','1','3')

insert into transactn values('2',5000,'2020-05-01','1','1')

insert into transactn values('3',15000,'2020-05-05','1','2')

insert into transactn values('4',20000,'2020-05-01','1','4')

insert into transactn values('5',30000,'2020-05-03','1','5')

select \* from transactn

alter table employee

add constraint ck1

foreign key (hr) references HR (hr\_id)

insert into funds values ('1','1',80000)

insert into salary values ('1','5','136195',30000,0,30000)

insert into salary values ('2','4','2438142',18000,2000,20000)

insert into salary values ('3','3','176351',15000,0,15000)

insert into salary values ('4','2','3379913',4500,500,5000)

insert into salary values ('5','1','1965216',9000,1000,10000)

select \* from employee

select \* from accountant

select \* from HR

select \* from transactn

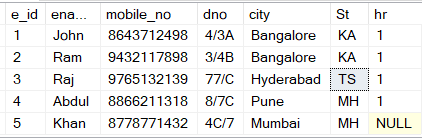
select \* from funds

select \* from salary

insert into employee values ('6','Kumar','9965684399','4A/C','Chennai','TN','1')

**DATABASE SCREENSHOTS**:

**employee**



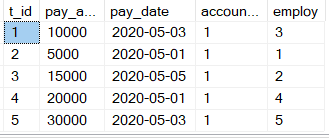
**accountant**



**HR**



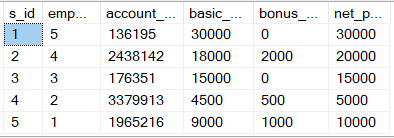
**transactn**



**funds**



**salary**



**SQL QUERIES:**

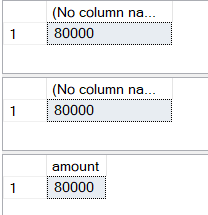
**1. Finding total amount of funds allocated and salaries given with reference to**

**Transactions**

SELECT SUM(pay\_amt) FROM transactn

SELECT SUM(net\_pay) FROM salary

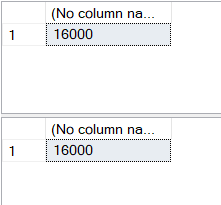
SELECT amount FROM funds



**2. Finding the average salary given to employees**

SELECT AVG(net\_pay) FROM salary

SELECT AVG(pay\_amt) FROM transactn

****

**3. Finding the maximum salary given**

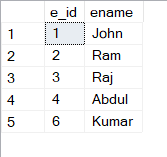
SELECT MAX(net\_pay) from salary



**4. Displaying employees under HR with id = 1**

SELECT e\_id,ename from employee where hr in

(SELECT hr\_id from HR where hr\_id = '1');



**5. Finding transaction id and employee id of employee with maximum salary**

SELECT t\_id, employ from transactn where pay\_amt in

(SELECT MAX(net\_pay) from salary);



**6. Displaying the employee’s details with their transaction ids using inner join**

SELECT

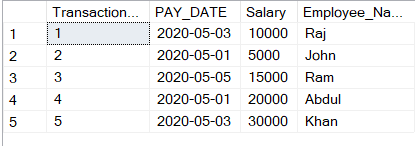
t2.t\_id as Transaction\_id, t2.pay\_date as PAY\_DATE, t2.pay\_amt as Salary,

t1.ename as Employee\_Name from

transactn t2

INNER JOIN employee t1

ON t2.employ like t1.e\_id



**7. Displaying the details of employees with their HR’s name**

SELECT

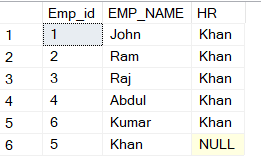
t2.e\_id as Emp\_id, t2.ename as EMP\_NAME, t1.hr\_name as HR

from

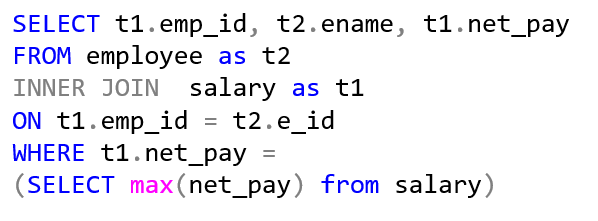
employee t2

FULL OUTER JOIN HR t1

ON t1.hr\_id like t2.hr



**8. Displaying the employee id, employee name and salary of the employee with highest salary (correlated – nested)**





**TRIGGERS**

A **trigger** is a special type of stored procedure that automatically runs when an event occurs in the database server. DML **triggers** run when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

**Creating a trigger**

**1. A trigger named “tg1” is created here that displays the whole contents of employee table whenever an HR adds a new employee to the database.**

create trigger tg1 on employee

after insert

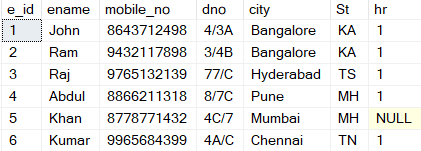
as

begin

select \* from employee

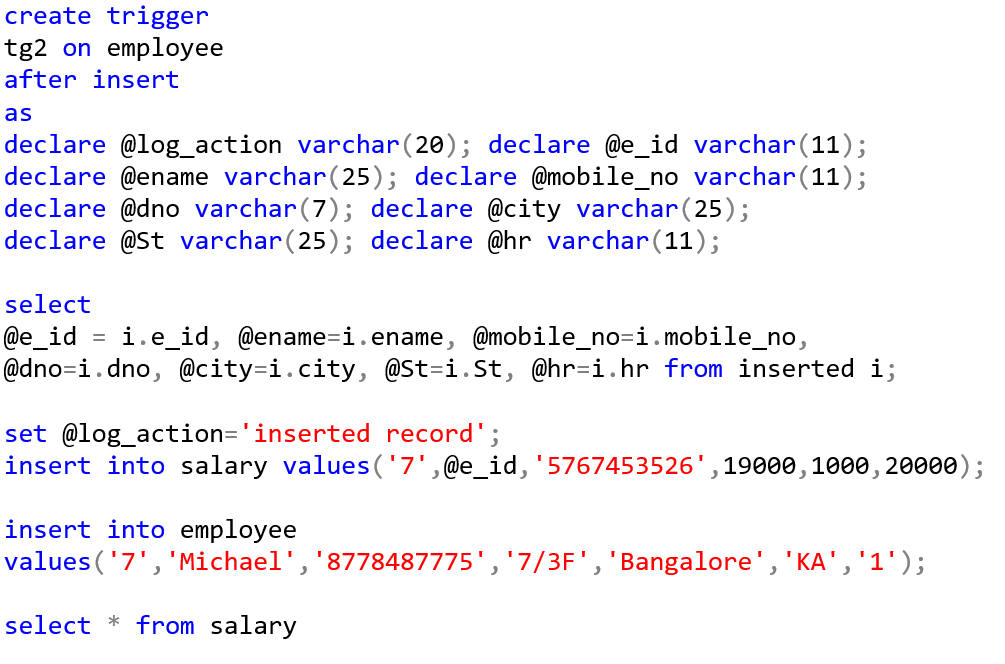
end

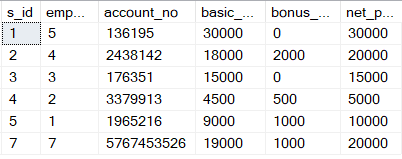
insert into employee values ('6','Kumar','9965684399','4A/C','Chennai','TN','1')



The above table is automatically displayed once a new employee details is added to the database (employee table).

**2. A trigger “tg2” is created such that whenever a new employee is added to the employee relation, his salary details gets updated automatically.**



****

**CONCLUSION**

In this project we have created an application which is easy to access and user friendly. The application keeps a backup of the Payroll (salary) management data which includes employee’s details such as salary, transaction, HR, funds allocated etc.

The implementations are successful with utmost care on constrains and checks.

Having a structured database like depicted in this project for salary management has a very good scope to keep track of each and every details so that even if there are discrepancies in any one table, those can be easily corrected with references from other tables.

**FUTURE ENHANCEMENT**

This database can be developed further to hold other details such as PF amount data, tax data etc. which helps in keeping track of complete employee details so that both the HRs and employees have a clear view on all their details.

Also frontend can be developed for this database to make the queries user-friendly

so that users can easily skim the data stored and retrieve the information easily. PHP and JavaFX can be used to develop the front end.