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

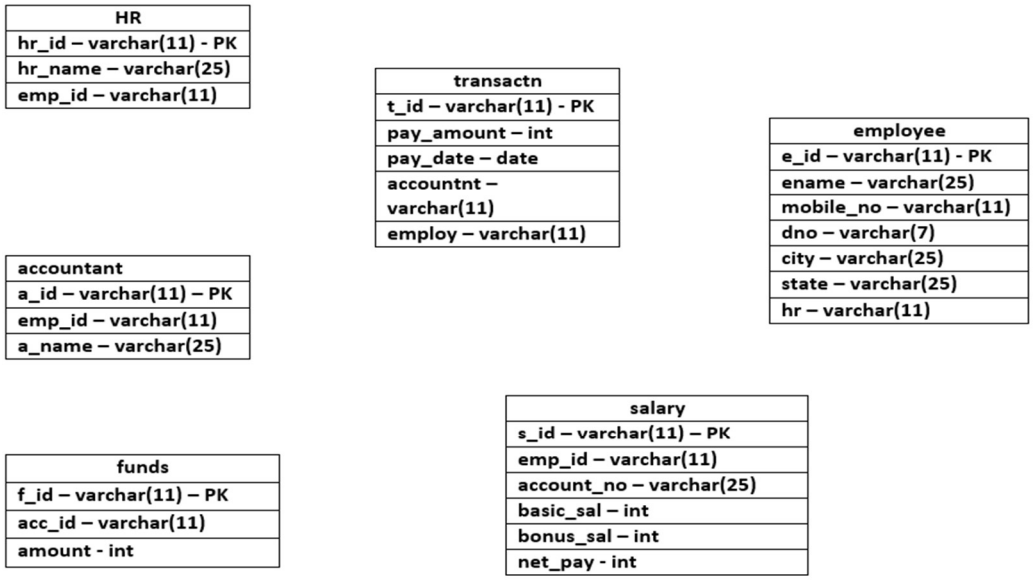
The main objective of this project is to demonstrate the way how payroll (salary) management is used in companies to assign salaries to their employees and also to mimic the process of allocation of funds to the employees.

There are five entities used here:

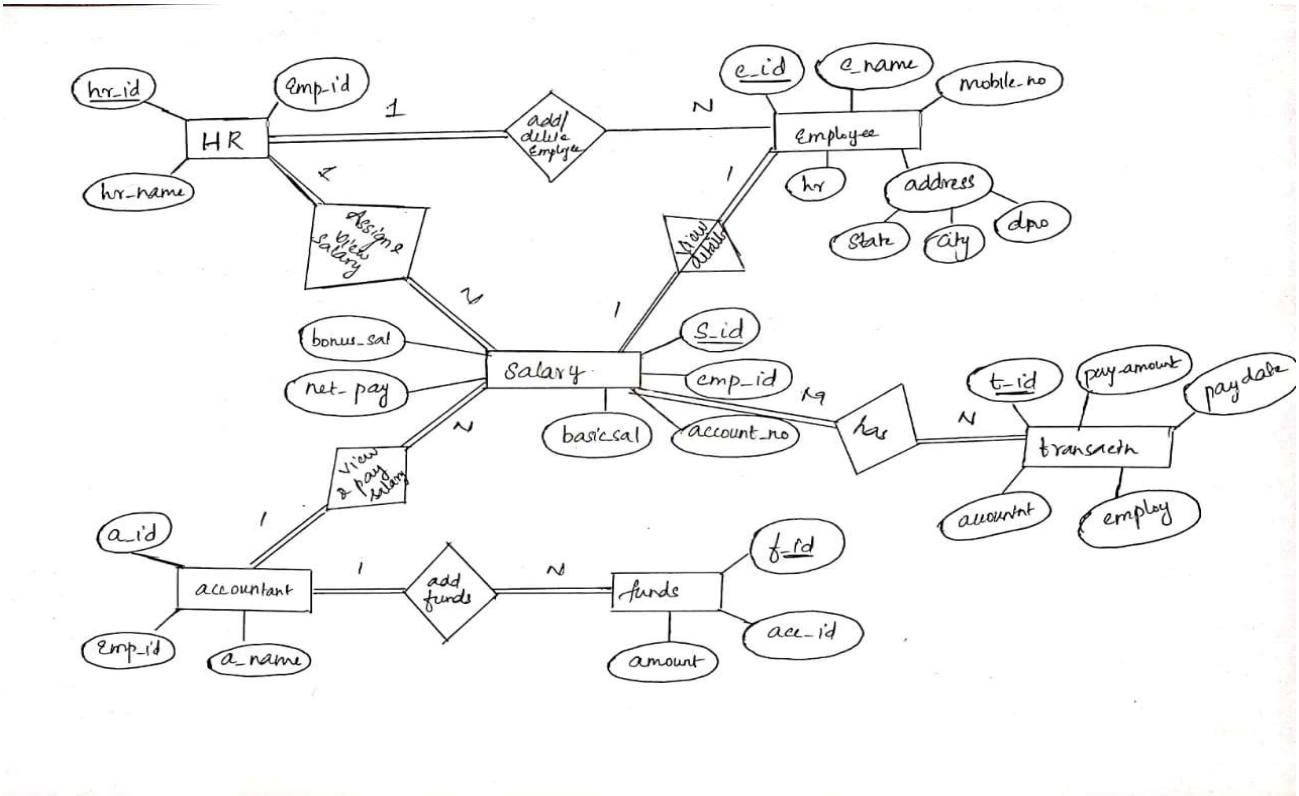
HR	add/delete employee assign/view salaries
ACCOUNTANT	view/pay salaries add funds
EMPLOYEE	view their salaries
SALARY	contains details of all salaries given also contains derived column - **transaction**
FUNDS	contains details about total funds allocated to assign salaries

DATA MODEL

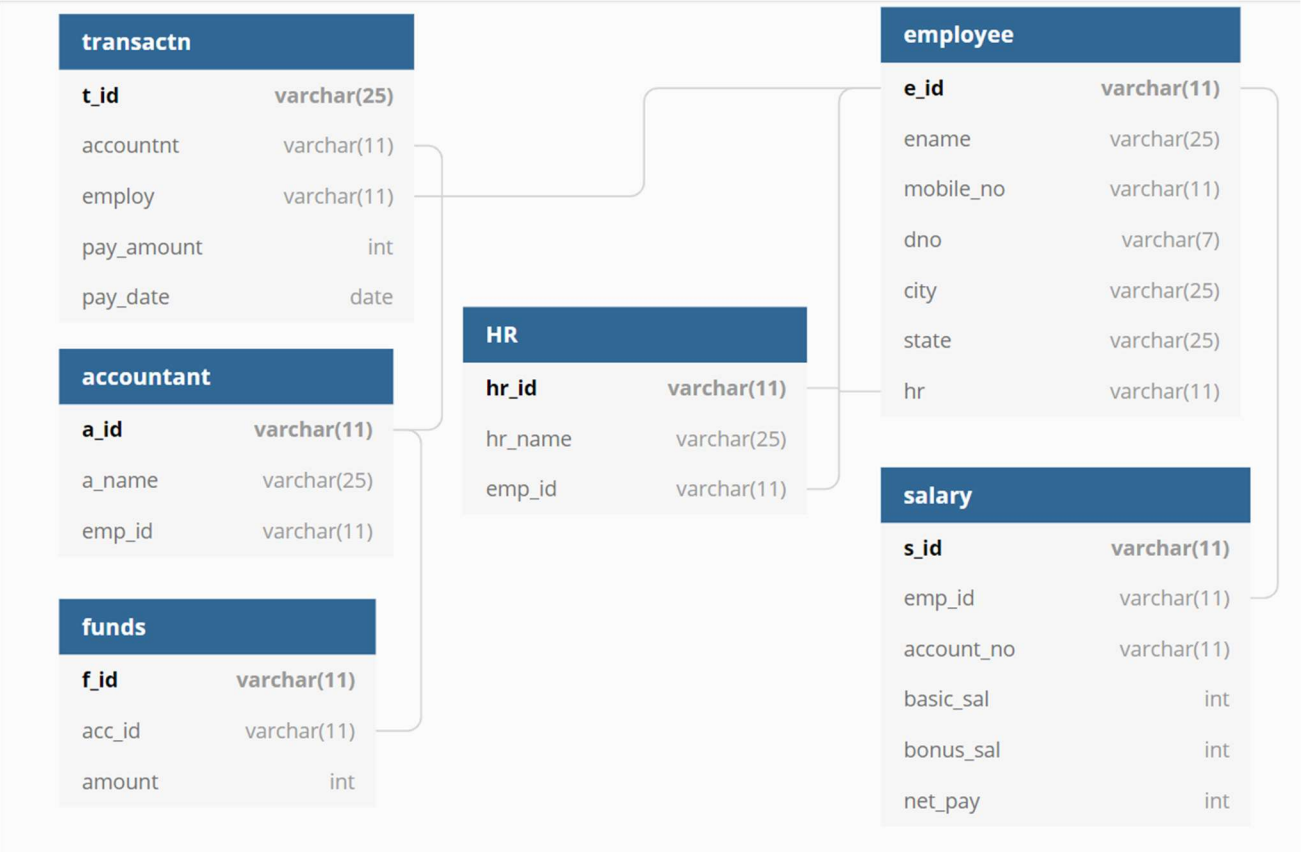
1. VARIABLE DESCRIPTION



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)

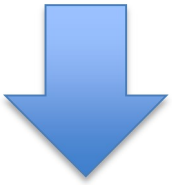
FD (s_id) -> {emp_id, account_no, basic_pay, bonus_pay, net_pay)

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

e_id	ena...	mobile_no	dno	city	St	hr
1	John	8643712498	4/3A	Bangalore	KA	1
2	Ram	9432117898	3/4B	Bangalore	KA	1
3	Raj	9765132139	77/C	Hyderabad	TS	1
4	Abdul	8866211318	8/7C	Pune	MH	1
5	Khan	8778771432	4C/7	Mumbai	MH	NULL

accountant

a_id	emp...	a_na...
1	3	Raj

HR

hr_id	hr_na...	emp_id
1	Khan	5

transactn

t_id	pay_a...	pay_date	accoun...	employ
1	10000	2020-05-03	1	3
2	5000	2020-05-01	1	1
3	15000	2020-05-05	1	2
4	20000	2020-05-01	1	4
5	30000	2020-05-03	1	5

funds

f_id	ac...	amo...
1	1	80000

salary

s_id	emp...	account_...	basic_...	bonus_...	net_p...
1	5	136195	30000	0	30000
2	4	2438142	18000	2000	20000
3	3	176351	15000	0	15000
4	2	3379913	4500	500	5000
5	1	1965216	9000	1000	10000

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
```

	(No column na...
1	80000

	(No column na...
1	80000

	amount
1	80000

2. Finding the average salary given to employees

```
SELECT AVG(net_pay) FROM salary
SELECT AVG(pay_amt) FROM transactn
```

	(No column na...
1	16000

	(No column na...
1	16000

3. Finding the maximum salary given

```
SELECT MAX(net_pay) from salary
```

	(No column na...
1	30000

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');
```

	e_id	ename
1	1	John
2	2	Ram
3	3	Raj
4	4	Abdul
5	6	Kumar

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);
```

	t_id	employ
1	5	5

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
```

	Transaction...	PAY_DATE	Salary	Employee_Na...
1	1	2020-05-03	10000	Raj
2	2	2020-05-01	5000	John
3	3	2020-05-05	15000	Ram
4	4	2020-05-01	20000	Abdul
5	5	2020-05-03	30000	Khan

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
```

	Emp_id	EMP_NAME	HR
1	1	John	Khan
2	2	Ram	Khan
3	3	Raj	Khan
4	4	Abdul	Khan
5	6	Kumar	Khan
6	5	Khan	NULL

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

```
SELECT t1.emp_id, t2.ename, t1.net_pay
FROM employee as t2
INNER JOIN  salary as t1
ON t1.emp_id = t2.e_id
WHERE t1.net_pay =
(SELECT max(net_pay) from salary)
```

emp_id	ena...	net_p...
5	Khan	30000

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')
```

e_id	ename	mobile_no	dno	city	St	hr
1	John	8643712498	4/3A	Bangalore	KA	1
2	Ram	9432117898	3/4B	Bangalore	KA	1
3	Raj	9765132139	77/C	Hyderabad	TS	1
4	Abdul	8866211318	8/7C	Pune	MH	1
5	Khan	8778771432	4C/7	Mumbai	MH	NULL
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.

```
create trigger
tg2 on employee
after insert
as
declare @log_action varchar(20); declare @e_id varchar(11);
declare @ename varchar(25); declare @mobile_no varchar(11);
declare @dno varchar(7); declare @city varchar(25);
declare @St varchar(25); declare @hr varchar(11);

select
@e_id = i.e_id, @ename=i.ename, @mobile_no=i.mobile_no,
@dno=i.dno, @city=i.city, @St=i.St, @hr=i.hr from inserted i;

set @log_action='inserted record';
insert into salary values('7',@e_id,'5767453526',19000,1000,20000);

insert into employee
values('7','Michael','8778487775','7/3F','Bangalore','KA','1');

select * from salary
```

s_id	emp...	account_no	basic_...	bonus_...	net_p...
1	5	136195	30000	0	30000
2	4	2438142	18000	2000	20000
3	3	176351	15000	0	15000
4	2	3379913	4500	500	5000
5	1	1965216	9000	1000	10000
7	7	5767453526	19000	1000	20000

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.