**PROJECT 2: GROUP 1**

**Akshay Venkataraman**

**Amandeep Kaur**

**Pavithra Thanikanti**

**Harneet Luthra**

**Gurjit Singh Khatkar**

**Manpreet Kaur**

**Joseph Charles**

**Rahul George**

**Pankaj Vats**

**Runze Li**

**Gurmeet**

**IT WORKS - Company Database Documentation**

**Company Overview:** IT works is a company specializing in designing databases. Following are some noteworthy points about IT works to enable the readers of this documentation to get a comprehensive overview of organizational structure and work culture of IT Works :

* IT Works comes under the category of companies having less than 100 employees.
* Also, it is flat company as there are no managers in this company. Company is comprised of the following types of employees:

-> 30 programmers.

-> 10 senior consultants.

-> 20 consultants.

-> Contractors.

* The other notable thing is that all employees work from home.
* Certain number of programmers, senior consultants, consultants are contractors are assigned to each project.
* Apparently, it is evident that this company is a project-driven company.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Why does IT Works need a database:** As mentioned above IT Works is an IT company specializing in databases. They are “database house” due to their expertise in designing large scale custom databases. Surprisingly, IT works does not have an in house database for their own use and they are operating on a paper based system, which can be error-prone. Keeping in view this problem, a database is required to automate the following tasks, with budgeting being the main focus:

* Keeping track of whether a project has more than enough people or less than enough people assigned to it.
* Making sure that all people work for designated time period during each day by introducing a login portal for employees.
* Keeping track of how much is being paid to employees on bench.
* Keeping track of hours worked by all employees ranging from programmers to consultants - as this is crucial for budgeting as well!
* Keeping track of the profit made for project(s) by using the following formula:

*Profit made is determined from the amount spent on employees working on that project subtracted from Project Budget.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

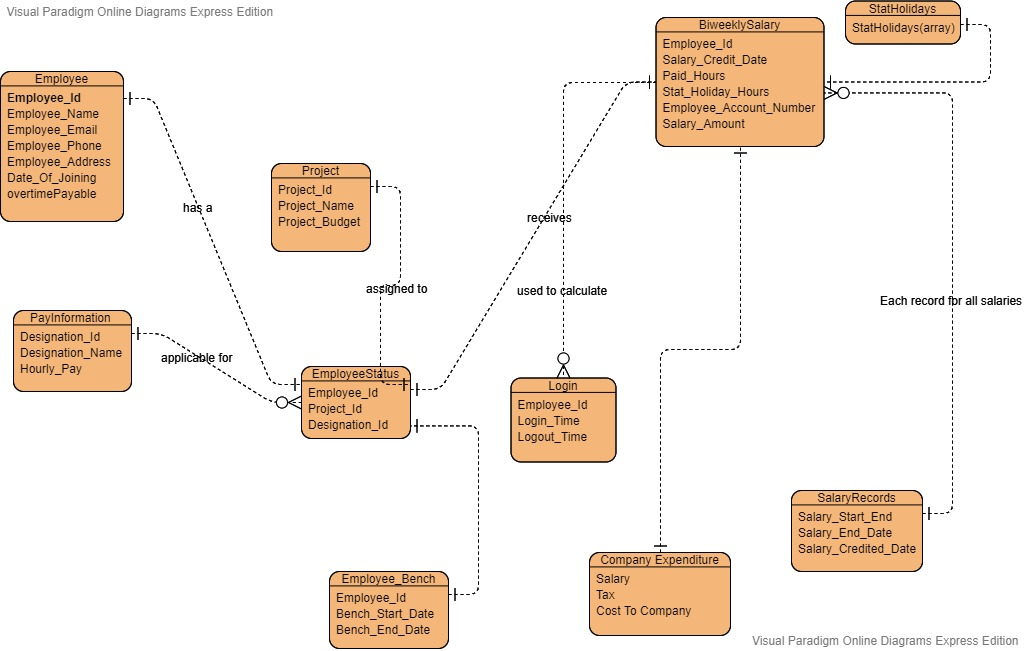
**Basic things to consider for designing the company database:** There are following constraints/conditions to be considered and incorporated into the database:

* Only contractors will get paid overtime.
* Employees are assigned to each project in the following manner:
* 1 Senior Consultant.
* 4 Consultants.
* 5 Programmers.
* 6 Contractors.
* Each employees are expected to work for 8 hours/day and 40 hours/week.
* Employees work from home but hours worked by them are tracked through employee login portal.
* IT Works has the concept of “On\_Bench”, which is applicable when there are not enough projects to work on.
* While benched, employees will still get paid whereas contractors will not get paid in this situation.
* For overtime, company is supposed to pay time and a half.
* If employees are benched then they are assigned to existing projects!
* On public holidays, employees will get paid double time, provided they work on stat holiday(s).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Overview of the database:** The following flowchart provides information about top level logical view of database in form an Entity Relationship Diagram which will be expanded on in later sections :

**Link to Database model:** <https://diagrams.visual-paradigm.com/#workspace=ueqexaqf&proj=0&id=1>



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Creation of Database:**

CREATE DATABASE IF NOT EXISTS BudgetingSystem;

USE BudgetingSystem;

**UNIQUE ENTITIES AND THEIR ATTRIBUTES**

**(Employee) - Table**

Employee\_Id: (int) Employee ID will be used as primary key in this table.

Employee\_Name: (String) Name of the Employee

Employee\_Email: (String) Official Email Id of the Employee used as contact

Employee\_phone: (String) Phone number used as the contact information

Employee\_Address: (String) Address of the Employee

Date\_Of\_Joining: (Date) Employee’s date of joining in the company

overtimePayable: (Boolean) Is a Boolean entity which will result in TRUE or FALSE if the Employee working in a project will be paid overtime or not. This boolean basically helps to determine if a person working on a project is contractor or not.

**Creation of Table:**

CREATE TABLE Employee (

employee\_id INT NOT NULL AUTO\_INCREMENT,

employee\_name VARCHAR(100) NOT NULL,

employee\_email VARCHAR(200) NOT NULL,

employee\_phone CHAR(10) NOT NULL,

employee\_address VARCHAR(100) NOT NULL,

date\_of\_joining DATE NOT NULL,

overtime\_payabale BOOL NOT NULL,

PRIMARY KEY(employee\_id)

);

**Populating dummy records:**

insert into Employee values(000001,'Akshay','akshay@gmail.com','7267687789','Mccallum','2018-11-10',1);

insert into Employee values(000002,'Rahul','rahul@gmail.com','7264560992','Kind Rd','2017-1-16',1);

insert into Employee values(000003,'Gurjit','gurjit@gmail.com','7264550225','UFV Rd','2015-10-11',1);

insert into Employee values(000004,'Ishaan','ishaan@gmail.com','7237562345','Mcullaum dr','2017-1-16',1);

insert into Employee values(000005,'Joseph','joseph@gmail.com','7269840938','Robson St','2017-1-16',1);

select \* from Employee;

**(Project) - Table**

Project\_Id: (int) Primary key in Project

table which will be used as Identity for the project

Project\_name: (string) The Name of the project

Project\_Budget: (int) Budget allocated for a project

**Creation of Table:**

CREATE TABLE Project (

project\_id INT NOT NULL AUTO\_INCREMENT,

project\_name VARCHAR(200) NOT NULL,

project\_budget DOUBLE NOT NULL,

PRIMARY KEY (project\_id)

);

**Populating dummy records:**

insert into Project values(5001,'PROJ1',25000);

insert into Project values(5002,'PROJ2',30010);

insert into Project values(5003,'PROJ3',98000);

insert into Project values(5004,'PROJ4',45789);

insert into Project values(5005,'PROJ5',65879);

select \* from Project;

**(EmployeeStatus) - Table**

Designation\_Id: (int) Employee designation will be allotted an ID which will be used to identify their pay scale

(Employee\_OnBench) - Table

Bench\_Start\_Date: (int) Will be used when an employee stops clocking in and the Bench date will start counting.

Bench\_Start\_Date: (int) Will be used when an employee stops clocking in and the Bench date will start counting.

Bench\_End\_Date: (int) Will end when the employee is allocated back in the project and he/she starts clocking in.

**Creation of Table:**

CREATE TABLE EmployeeStatus(

employee\_id INT NOT NULL,

project\_id INT,

designation\_id INT NOT NULL,

PRIMARY KEY(employee\_id)

);

**Populating dummy records:**

insert into EmployeeStatus values(000001,,3001);

insert into EmployeeStatus values(000002,5002,3002);

insert into EmployeeStatus values(000003,5003,3003);

insert into EmployeeStatus values(000004,5004,3004);

insert into EmployeeStatus values(000005,5005,3005);

select \* from EmployeeStatus;

**(StatHolidays) - Table**

Stat\_Holidays: (array) This array will have list of Stat Holidays

**Creation of Table:**

CREATE TABLE StatHolidays (

stat\_holiday\_date DATE

);

**Populating dummy records:**

insert into StatHolidays values('2019-01-01');

insert into StatHolidays values('2019-02-18');

insert into StatHolidays values('2019-04-19');

insert into StatHolidays values('2019-04-22');

insert into StatHolidays values('2019-05-20');

select \* from StatHolidays;

**(Login) - Table**

Login\_Time: (time) Time when the employee logs in

Logout\_Time: (time) Time when the employee logs out

Total\_Hours: (time) Hours and minutes between Login and Logout time

**Creation of Table:**

CREATE TABLE Login (

employee\_id INT NOT NULL,

login\_time DATETIME,

logout\_time DATETIME,

PRIMARY KEY(employee\_id)

);

**Populating dummy records:**

insert into Login values(000001,'2019-09-29 8:00:17','2019-09-29 17:00:15');

insert into Login values(000002,'2019-09-29 8:00:20','2019-09-29 18:00:30');

insert into Login values(000003,'2019-09-29 8:00:11','2019-09-29 16:00:40');

insert into Login values(000004,'2019-09-29 8:00:12','2019-09-29 20:00:45');

insert into Login values(000005,'2019-09-29 8:00:09','2019-09-29 19:00:32');

select \* from Login;

**(BiweeklySalary) - Table**

Paid\_Hours: (float) Total hours the employee worked

Stat\_Holiday\_Hours: (float) Total hours of Stat holiday

Salary\_Credited\_Date: (Date) Date the salary was credited to employee bank acc.

Salary\_Amount: (float) Total salary amount based on hours worked in two weeks’ time, overtime if applicable and stat holiday

**Creation of Table:**

CREATE TABLE BiweeklySalary(

employee\_id INT NOT NULL,

salary\_credited\_date DATE,

paid\_hours INT,

stat\_holiday\_hours INT,

employee\_account\_number VARCHAR(100),

salary\_amount FLOAT,

PRIMARY KEY(employee\_id)

);

**Populating dummy records:**

insert into BiweeklySalary values(000001,'2019-01-14',72,8,'1002612401',10000);

insert into BiweeklySalary values(000002,'2019-01-14',80,8,'1002612402',8000);

insert into BiweeklySalary values(000003,'2019-01-14',80,8,'1002612403',9000);

insert into BiweeklySalary values(000004,'2019-01-14',80,8,'1002612405',6000);

insert into BiweeklySalary values(000005,'2019-01-14',75,8,'1002612406',7000);

select \* from BiweeklySalary;

**(SalaryRecords) - Table**

Salary\_End\_Date: (Date) Saturday of 2nd week will the salary end date

Salary\_Credited: (Date) The Date Salary will be credited bi-weekly

**Creation of Table:**

CREATE TABLE SalaryRecords(

employee\_id INT NOT NULL,

salary\_start\_date DATE,

salary\_end\_date DATE,

PRIMARY KEY (employee\_id)

);

**Populating dummy records:**

insert into SalaryRecords values(000001,'2019-01-01','2019-01-14');

insert into SalaryRecords values(000002,'2019-01-01','2019-01-14');

insert into SalaryRecords values(000003,'2019-01-01','2019-01-14');

insert into SalaryRecords values(000004,'2019-01-01','2019-01-14');

insert into SalaryRecords values(000005,'2019-01-01','2019-01-14');

select \* from SalaryRecords

**(PayInformation) - Table**

Designation\_Id: (int) Id assigned to each designation type

Designation\_Name: (string) Name given to the designation

Hourly\_Pay: (float) Pay classification as per designation

**Creation of Table:**

CREATE TABLE PayInformation (

designation\_id INT NOT NULL,

designation\_name VARCHAR(50) NOT NULL,

hourly\_pay DOUBLE NOT NULL,

PRIMARY KEY (designation\_id)

);

**Populating dummy records:**

insert into PayInformation values(3001,'Programmers',20.5);

insert into PayInformation values(3002,'Senior consultant',32.5);

insert into PayInformation values(3003,'Consultant',25.2);

insert into PayInformation values(3004,'Contractors',28.0);

CREATE TABLE CompanyExpenditure (

salary FLOAT,

tax FLOAT,

cost\_to\_company FLOAT

);

insert into CompanyExpenditure values(10000,1200,11200);

insert into CompanyExpenditure values(8000,1212,9212);

insert into CompanyExpenditure values(9000,1500,10500);

insert into CompanyExpenditure values(6000,922,6922);

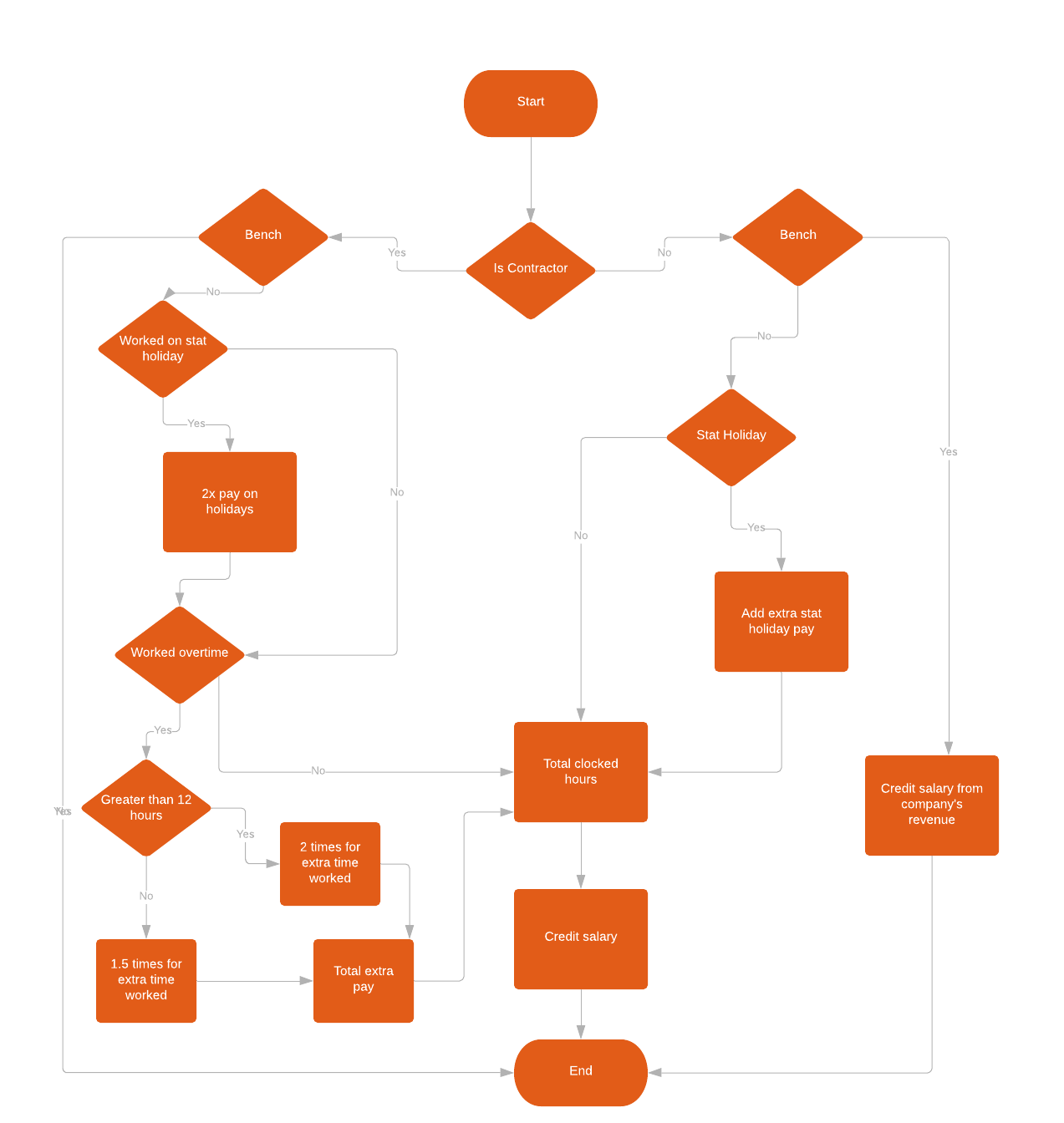
insert into CompanyExpenditure values(7000,980,7980);

select \* from CompanyExpenditure;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Detailed logic implemented in database:--**

As IT works, is project based company it is important to know about how different categories of employees are paid resulting in assurance of the fact that project budget is utilised properly within company’s limits. There are different rules for paying employees and contractors for overtime as well as stat holiday pay, which is depicted by the flowchart given below:

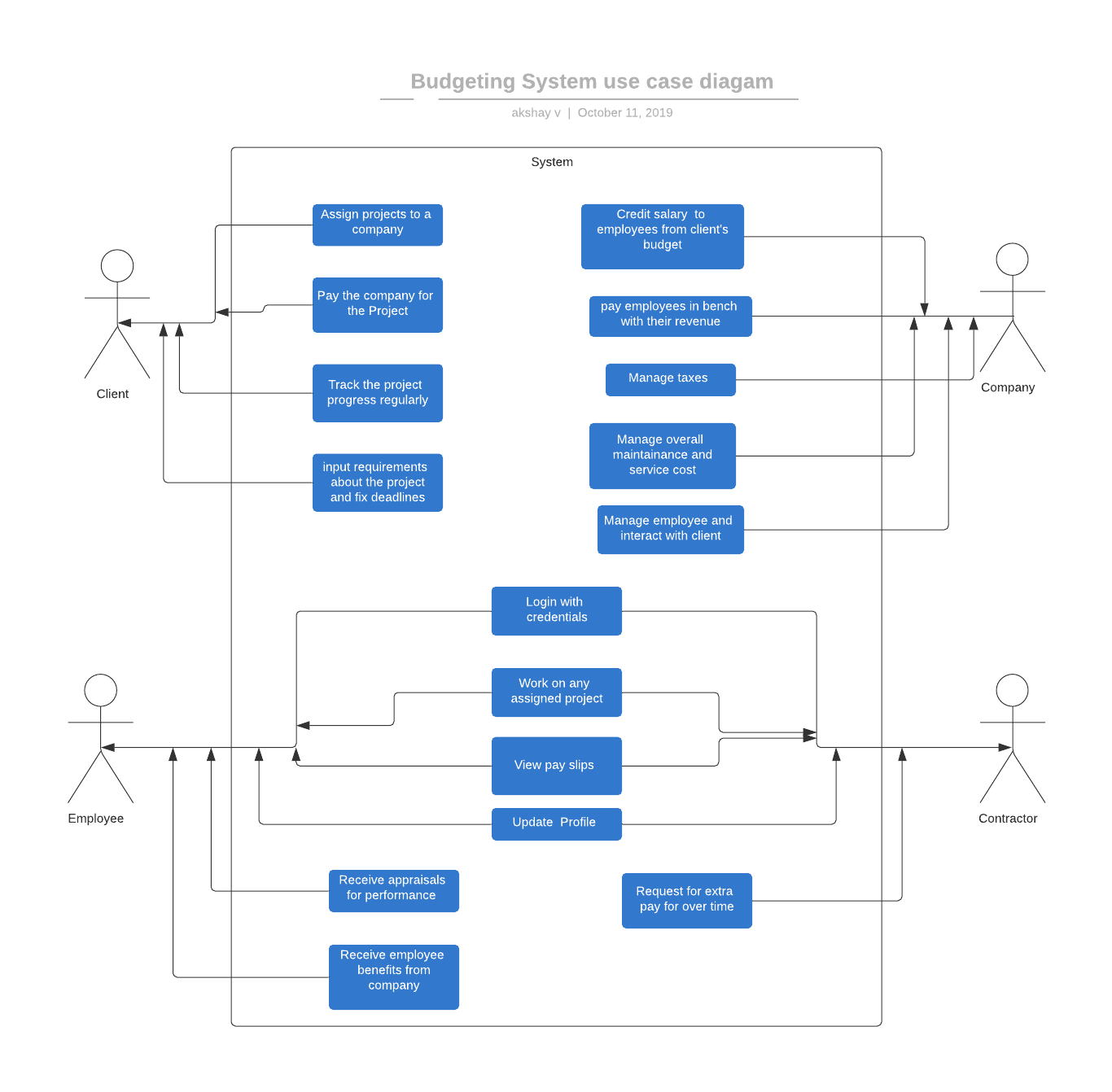
****

Above flowcharts starts from checking the condition if the employee is a contractor or not. If the condition is yes next step will be a condition where the contractor is benched or not. If yes the flowchart will move to total clocked hours. If not the flow chart will end. If not benched then it again checks condition if an employee worked more than eight hour. If it is below 8 hours basic pay scale will be exercised. If it is above 8 hour it will again check if its below 12 or above 12 hour. In case of 12 hour additional 4 hours will be paid by 1.5 times. If it's above 12 additional hours will be paid by 2 times.

Going back to the (if contractor or not condition) and if the condition is no then it will check the stat holiday condition. If it is not a stat holiday it will flow to total clocked hours. If Stat holiday condition is true then it will check for condition if the employee was working or not. If yes then employee will get double pay on total hours worked. If not, employee will be paid for 8 hours like a regular pay even if the employee doesn’t clock in. Then the total additional pay will be added with total clocked hours. Then the salary will be credited. And that's the end of flow chart

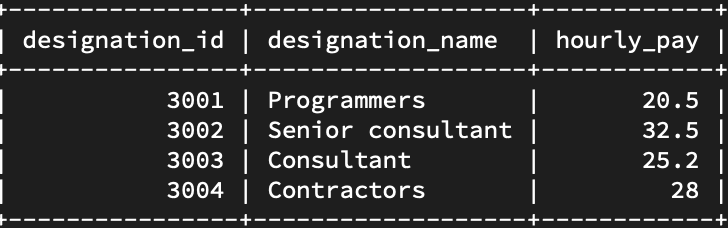
**Budgeting System Case Diagram**

Below diagram explain how four main components in the project interact in the system. Clients have projects that are assigned to the company which will incur certain cost and time for the company to work with. Meanwhile clients can track the progress of the project and give feedback or request changes as the work is in progress. On the other side company manages employees who are assigned a project and even pay the ones who are on bench. Company is also responsible for the maintenance of the system. Another challenge is managing employees and contractors as they both have different set of rules where a contractor gets paid for overtime while the employee is not. Employees get paid stat holiday while contractors don't and other things like appraisals can be managed.

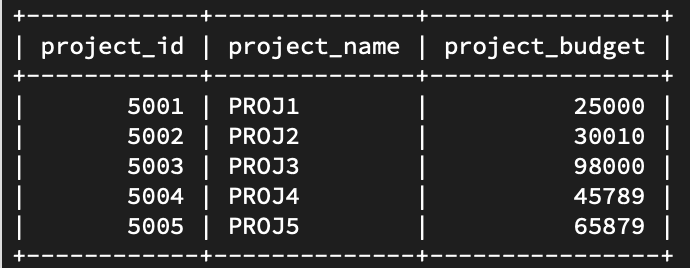


**Views implemented for the budgeting system:**

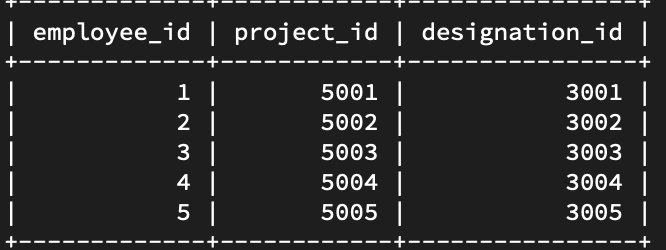
1. **Showing detailed information about designation and respective pay rates:** We implemented a view to show information about designation IDs, designation names, and hourly pay rates for employees within the company. The following output is given by this view and it shows required information about pay rates for employees with different designation names. This view has necessary information whenever payroll wants to double check biweekly payments made to different employees associated with different designations.

****

**2. Showing detailed information about existing projects:**Often it is required to obtain an overview on ongoing projects within the company. Keeping in view this idea, we implemented a view to show information about current projects running the company , and then information from this view can be used for doing further queries about the projects.



**3. Information about employees associated with a particular project(s):**It is required to show information about which employees are working on which projects. Following output of a code snippet gives required information about this:



**GroupBy:**

The GROUP BY statement groups rows that have the same values into summary rows.

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

For instance, I need to see count of employees assigned to a single project. Below query gives number of employees for each project.

SELECT COUNT(employee\_id ), project\_id

FROM EmployeeStatus

GROUP BY project\_id ;

**Having:**

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Now, I can use the below query to retrieve all the project\_id’s that have more than 10 employees assigned.

SELECT COUNT(employee\_id ), project\_id

FROM EmployeeStatus

GROUP BY project\_id

Having COUNT(employee\_id ) >10 ;

**Procedures:**

We introduced some basic procedures to reuse the code and understand the functionality and execution of advanced objects other than tables.

1. Created a procedure to retrieve all active employees in a system. Below is the code to create procedure. This pulls all records from EmployeeStatus table where the project\_id is null including Employee Name, ID, Phone and Project Name.

**Creation of Procedure:**

CREATE PROCEDURE GetallActiveEmployees

AS

Select

a.employee\_name,

a.employee\_id,

a.employee\_phone,

Projectname as (select c.project\_name from Project c where c.project\_id= b.project\_id)

from

Employee a , EmployeeStatus b

where a.employee\_id = b.employee\_id

and b.project\_id is not null;

GO;

**Execution of Procedure:**

EXEC GetallActiveEmployees;

1. We created a procedure to retrieve all the overtime employees.

This pulls data for all employees where (login\_time-logout\_time) >8 hrs.

**Creation of Procedure:**

CREATE PROCEDURE GetallOvertimeEmployees

AS

Select

a.employee\_name,

a.employee\_id,

Projectname as (select c.project\_name from Project c where c.project\_id= b.project\_id)

from

Employee a , EmployeeStatus b, Login l

where a.employee\_id = b.employee\_id

and l.employee\_id = a.employee\_id

and l.employee\_id = b.employee\_id

and (l.logout\_time - l.login\_time) > 8

and b.project\_id is not null;

GO;

**Execution of Procedure:**

EXEC GetallOvertimeEmployees;

**Procedure using UPDATE:**

If we have to update any information in tables, we use ‘UPDATE’ key word to do it in database.

For instance, I want to update email address for an employee in Employee table.

Update Employee set employee\_email = ‘joseph12@gmail.com’where employee\_id=000005 ;

Commit;

Note: If we do not use commit at the end, then the update we did will not be completed and will not be updated in database.

To make this easier, we can always use this update statement in a procedure and just call procedure whenever you want to update email address for an employee.

**Creation of Procedure:**

CREATE PROCEDURE UpdateEmailaddress @email varchar(200), @employeeid int

AS

Update Employee set employee\_email = @email

where employee\_id= @employeeid ;

Commit;

GO;

**Execution of Procedure:**

EXEC UpdateEmailaddress @email =’[joseph12@gmail.com](mailto:joseph12@gmail.com)’ ,@employeeid=000005

1. select queries: includes function like average, sum , min, max, count, order by, desc, asc, and,like, distinct, group by,not in, Between

Union all, union;

select \* from Company\_Expenditure UNION ALL Project;

select Salary from Company\_Expenditure UNION select Project\_Name from Project order by Project\_id;

This query selects all employees and groups them based on their date of joining. Basically, this query displays various distinct dates on which company hired at least one employee:

select Employee\_id, Employee\_Name from Employee GROUP BY Date\_Of\_Joining;

This query selects all Projects but not those projects which have employees associated having ids equal to 001 or 002 or 010

select \* from Project where Employee\_id NOT IN (001,002,010);

This query selects all salaries from Company Expenditure between $2000 and $4000

select \* from Company\_Expenditure where Salary BETWEEN 2000 AND 4000;

This query selects all employees from the employee table:

select \* from Employee;

This query finds name of employe whose employee\_id is 1.

select Employee\_Name from Employee where Employee\_id=1;

This query basically determines if employee with id=1 has phone number ending with the digit 1.

select Employee\_Name and Date\_Of\_Joining from Employee where Employee\_id=1and Employee\_Phone like '%1';

The following query selects all projects from the company database:

*select \* from Project;*

The following query selects number of projects currently running the company:

*select count(\*) from Project;*

Cost To Company holds values of various expenditures for the company and this query sums up values in Cost To Company column to give up to date cost incurred:

*select sum(Cost To Company) from Company Expenditure;*

The following query selects the average salary paid to employees:

*select AVG(Salary) from Company Expenditure;*

This query selects the minimum amount paid to any employee during the current pay period:

*select min(Salary) from Company Expenditure;*

This query selects the maximum amount paid to any employee during the current pay period:

*select max(Salary) from Company Expenditure;*

This query selects all employees from the databases but it only outputs unique occurences:

*select DISTINCT Employee\_Name from Employee;*

This query displays designation\_id for designations for which hourly pay is greater than 30 dollars

*select Designation\_Id from PayInformation where Hourly\_Pay>30;*

This query displays designation\_id for designations for which hourly pay is greater than 30 dollars but it selects only the top 5 entries or designation\_ids:

*select Designation\_Id from PayInformation where Hourly\_Pay>50 limit 5;*

This query sorts data in PayInformation in descending order based on first column:

*select \* from PayInformation order by 1 desc*;

This query sorts data in PayInformation in ascending order based on first column:

*select \* from PayInformation order by 1 asc;*

2. Update queries: update with greater than, equal to, updating 2 columns

This updates paid hours to 10 in BiweeklySalary table whichever rows have value of salary equals to 3000:

Update BiweeklySalary set paid\_Hours=10 where salary\_Amount=3000;

This updates paid hours to 20 in BiweeklySalary table whichever rows have value of salary greater than 3000:

Update BiweeklySalary set paid\_Hours=20 where salary\_Amount>=3000;

This is basically an example of query to run whenever we want to assign project to an employee on the bench:

Update EmployeeStatus set Project\_id=100 where Designation\_Id IS NULL;

This is an example query for updating the information about an existing project with new parameters/values:

Update Project set Project\_Name=asoss , Project\_id=101 where Designation\_Id=002;

3. Joins:

inner join

This join basically gets information from Employee table and EmployeeStatus table to display detailed information about on bench employees:

select Employee.Employee\_Name, Employee.Employee\_Phone, Employee\_Bench.Bench\_Start\_Date from Employee inner join Employee\_Bench on

Employee.Employee\_Id = Employee\_Bench.Employee\_Id;

left join

select Employee.Employee\_Name, Employee.Employee\_Email, Login.Login\_Time

from Employee left join Login on Employee.Employee\_Id= Login.Employee\_Id;

right join

select EmployeeStatus.Employee\_id, EmployeeStatus.Project\_id, Employee.Employee\_Name from EmployeeStatus right join Employee on EmployeeStatus.Employee\_id=Employee.Employee\_

id;

3. Delete:

This is an example how to delete an employee with a specific name from the Employee table:

delete from Employee where Employee\_Name=aman;

4. Alter:

This query drops the overtimePayable field in Employee table:

Alter table Employee DROP INDEX overtimePayable;

This query adds the overtimePayable field in Employee table:

Alter table Employee ADD CONSTRAINT overtimePayable;

This query adds employee\_id as a primary key field to Employee table:

Alter table Employee ADD PRIMARY KEY(Employee\_Id);

5. Case:

This is an example of advanced nested subquery. This query first executes the subquery within the parentheses which basically returns employee\_name if employee\_id is null otherwise it returns employee\_email and then outer statement/query displays data ordered by the value return from inner subquery.

select Employee\_id , Employee\_Email,Employee\_name from Employee Order by

(

Case

when Employee\_id IS NULL then Employee\_name

else Employee\_email

end

);

6. Nested queries/Subqueries:

This nested query first selects maximum value for CostToCompany field, then selects sum of taxes from that maximum value and selects salary for which maximum tax was deducted from Company\_Expenditure table

select \* from Company\_Expenditure where salary=(select sum(tax) from company\_Expenditure where Cost To Company=(select max(Cost To Company) from Company\_Expenditure));