# Table of Contents

# Assumptions

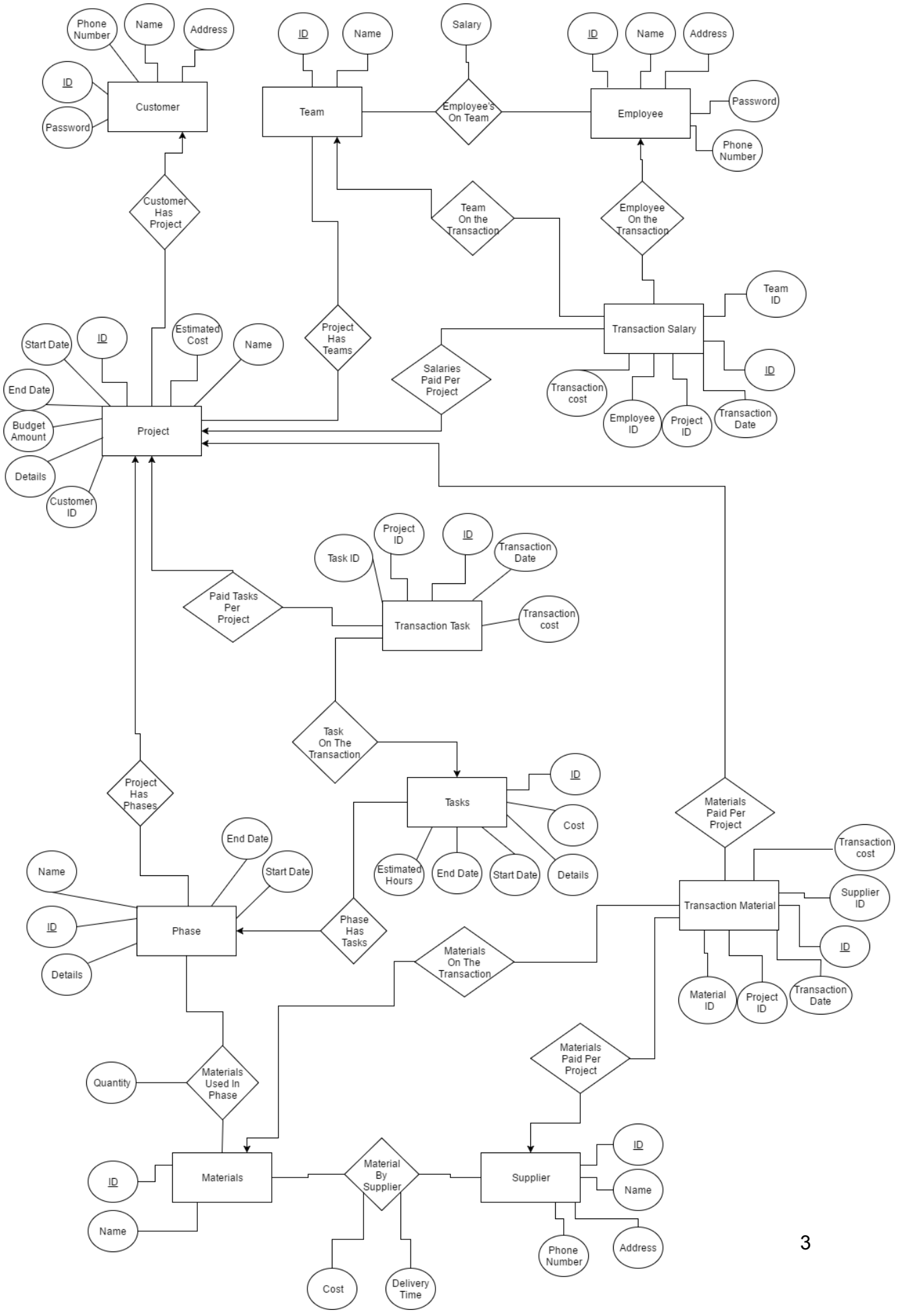
- Employees can be part of multiple teams
- Employee's monthly salary is defined by a team where he is working
- A team can be part of multiple projects
- Customers can have multiple projects
- The same material can come from different suppliers which define the material's price and delivery time
- Only one phase at a time can be in progress in a project

# Entity Relationship Diagram

https://www.draw.io/?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1&title=E%2FR#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D0B-SikSxC1cLBbWJqV0ZPWjZUdDA%26export%3Ddownload

Or a shortened link:

https://goo.gl/ylpBB1


Phone Number
Name
Address
ID
Password
Customer
ID
Name
Salary
Team
Employee's On Team
ID
Name
Address
Employee
Password
Phone Number
Customer Has Project
Team On the Transaction
Employee On the Transaction
Team ID
Transaction Salary
ID
Transaction cost
Employee ID
Project ID
Transaction Date
Start Date
ID
Estimated Cost
Name
End Date
Budget Amount
Details
Customer ID
Project
Project Has Teams
Salaries Paid Per Project
Project ID
ID
Task ID
Transaction Date
Paid Tasks Per Project
Transaction Task
Transaction cost
Task On The Transaction
Project Has Phases
ID
Tasks
Cost
Estimated Hours
End Date
Start Date
Details
Materials Paid Per Project
Transaction cost
End Date
Start Date
Name
ID
Details
Phase
Phase Has Tasks
Transaction Material
Supplier ID
ID
Materials On The Transaction
Material ID
Project ID
Transaction Date
Quantity
Materials Used In Phase
Materials Paid Per Project
ID
Name
Materials
Material By Supplier
Supplier
ID
Name
Address
Phone Number
Cost
Delivery Time

# Relational Schemas

Below you can find the conversion of the E/R diagram to relational schema, along with its constraints, primary keys and foreign keys.

**Customer** (custID, custName, custPwd, custAddress, custPhoneNum)

1. custID is set to: not null unique unsigned auto_increment
2. custName is set to: not null
3. custPwd is set to: not null
4. primary key (custID)

**Employee** (empID, empName, empPwd, empAddress, empPhoneNum)

1. empID is set to: unsigned not null unique auto_increment
2. empName is set to: not null
3. empPwd is set to: not null
4. primary key (empID)

**Team** (teamID, teamName)

1. teamID is set to: unsigned not null unique auto_increment
2. teamName is set to: not null
3. primary key (teamID)

**Team_Employee** (empID, teamID, salary)

1. empID is set to: unsigned not null
2. teamID is set to: unsigned not null
3. salary is set to: unsigned not null
4. primary key (empID, teamID)
5. foreign key (empID) references EMPLOYEE (empID),
6. foreign key (teamID) references TEAM (teamID)

**Project** (projID, custID, projName, projDetails, budgetAmount, estimatedCost, projDateStart, projDateEnd)

1. projID is set to: not null unique auto_increment
2. projName is set to: not null

3. budgetAmount is set to: unsigned
4. estimatedCost is set to: unsigned
5. primary key (projID)
6. foreign key (custID) references CUSTOMER (custID)

**Project_Team** (projID, teamID)

1. projID is set to: unsigned not null
2. teamID is set to: unsigned not null
3. primary key (projID, teamID)
4. foreign key (projID) references PROJECT (projID)
5. foreign key (teamID) references TEAM (teamID)

**Phase** (phaseID, projID, phaseName, phaseDetails, phaseDateStart, phaseDateEnd)

1. phaseID is set to: unsigned not null unique auto_increment
2. projID is set to: unsigned not null
3. phaseName is set to: not null
4. primary key (phaseID)
5. foreign key (projID) references PROJECT (projID)

**Material** (matID, matName)

1. matID is set to: unsigned not null unique auto_increment
2. primary key (matID)

**Phase_Material** (phaseID, matID, qty)

1. phaseID is set to: unsigned not null
2. matID is set to: unsigned not null
3. qty is set to: unsigned
4. primary key (phaseID, matID)
5. foreign key (phaseID) references PHASE (phaseID)
6. foreign key (matID) references MATERIAL (matID)

**Task** (taskID, phaseID, taskDetails, taskCost, taskEstimateHours, taskDateStart, taskDateEnd)

1. taskID is set to: unsigned not null unique auto_increment
2. phaseID is set to: unsigned not null
3. taskCost is set to: unsigned

4. taskEstimateHours is set to: unsigned not null
5. primary key (taskID)
6. foreign key (phaseID) references PHASE(phaseID)

**Supplier** (supID, supName, supAddress, supPhoneNum)

1. supID is set to: unsigned not null unique auto_increment
2. primary key (supID)

**Material_Supplier** (matID, supID, matCost, deliveryTime)

1. matID is set to: unsigned not null
2. supID is set to: unsigned not null
3. matCost is set to: unsigned
4. deliveryTime is set to: unsigned
5. primary key (matID, supID)
6. foreign key (matID) references MATERIAL (matID)
7. foreign key (supID) references SUPPLIER (supID)

**Transaction_Task** (transID, projID, taskID, transCost, transDate);

1. transID is set to: unsigned not null unique auto_increment
2. projID is set to: unsigned not null
3. taskID is set to: unsigned not null
4. transCost is set to: unsigned not null
5. transDate is set to: not null
6. primary key (transID)
7. foreign key (projID) references PROJECT (projID)
8. foreign key (taskID) references TASK (taskID)

**Transaction_Salary** (transID, projID, empID, teamID, transCost, transDate)

1. transID is set to: unsigned not null unique auto_increment
2. projID is set to: unsigned not null
3. empID is set to: unsigned not null
4. teamID is set to: unsigned not null
5. transCost is set to: unsigned not null
6. transDate is set to: not null
7. primary key (transID)
8. foreign key (projID) references PROJECT (projID)
9. foreign key (empID) references EMPLOYEE (empID)

10. foreign key (teamID) references TEAM (teamID)

**Transaction_Material** (transID, projID, matID, supID, transCost, transDate)

1. transID is set to: unsigned not null unique auto_increment
2. projID is set to: unsigned not null
3. matID is set to: unsigned not null
4. supID is set to: unsigned not null
5. transCost is set to: unsigned not null
6. transDate is set to: not null
7. primary key (transID)
8. foreign key (projID) references PROJECT (projID)
9. foreign key (matID) references MATERIAL (matID)
10. foreign key (supID) references SUPPLIER (supID)

# Functional Dependencies and Closures

**Customer** (custID, custName, custPwd, custAddress, custPhoneNum)

custID → custName, custPwd, custAddress, custPhoneNum
$\{custID\}^+$ = {custID,custName, custPwd, custAddress, custPhoneNum}

**Employee** (empID, empName, empPwd, empAddress, empPhoneNum)

empID → empName, empPwd, empAddress, empPhoneNum
$\{empID\}^+$ = {empID, empName, empPwd, empAddress, empPhoneNum}

**Team** (teamID, teamName)

teamID → teamName
$\{teamID\}^+$ = {teamID, teamName}

**Team_Employee** (empID, teamID, salary)

empID, teamID → salary
$\{empID, teamID\}^+$ = {empID, teamID, salary}

**Project** (projID, custID, projName, projDetails, budgetAmount, estimatedCost, projDateStart, projDateEnd)

projID → custID, projDetails, budgetAmount, estimatedCost, projDateStart, projDateEnd
$\{projID\}^+$ = {projID, custID, projDetails, budgetAmount, estimatedCost, projDateStart, projDateEnd}

**Project_Team** (projID, teamID)

No non trivial functional dependencies

**Phase** (phaseID, projID, phaseName, phaseDetails, phaseDateStart, phaseDateEnd)

phaseID, projID → phaseName, phaseDetails, phaseDateStart, phaseDateEnd

{phaseID, projID}$^+$ = {phaseID, projID, projDetails, budgetAmount, estimatedCost, projDateStart, projDateEnd}

**Material** (matID, matName)

matID → matName
{matID}$^+$ = {matID, matName}

**Phase_Material** (phaseID, matID, qty)

phaseID, matID → qty

**Task** (taskID, phaseID, taskDetails, taskCost, taskEstimateHours, taskDateStart date, taskDateEnd)

taskID, phaseID → taskDetails, taskCost, taskEstimateHours, taskDateStart date, taskDateEnd
{taskID, phaseID}$^+$ = {taskID, phaseID,taskDetails, taskCost, taskEstimateHours, taskDateStart date, taskDateEnd}

**Supplier**(supID, supName, supAddress, supPhoneNum)

supID → supName, supAddress, supPhoneNum
{supID}$^+$ = {supID, supName, supAddress, supPhoneNum}

**Material_Supplier** (matID, supID, matCost, deliveryTime)

matID, supID → matCost, deliveryTime
{matID, supID}$^+$ = {matID, supID, matCost, deliveryTime}

**Transaction_Task** (transID, projID, taskID, transCost, transDate);

transID → projID, taskID, transCost, transDate
{transID}$^+$ = {transID, projID, taskID, transCost, transDate }

**Transaction_Salary** (transID, projID, empID, teamID, transCost, transDate)

transID → projID, empID, teamID, transCost, transDate
{transID}$^+$ = {transID,  projID, empID, teamID, transCost, transDate }

**Transaction_Material** (transID, projID, matID, supID, transCost, transDate)

transID → projID, matID, supID, transCost, transDate
$\{transID\}^+$ = {transID, projID, matID, supID, transCost, transDate }

Our relations are all in BCNF because the left hand side of every relation's FD is a superkey.

# Design Details

The customer relation has an ID and a password which they will use to login to the website. A customer can have multiple projects but a project has only one customer.

The project relation has an estimated cost attribute and a budget amount attribute. The estimate cost is an estimation done by an employee to have a rough estimate of the total cost of the house. The budget amount represents the amount of money that is available from the invested funds of the client. When a project is first created, they invest an initial budget amount, and as the project progresses, they can top up the remaining budget. A project will always have a start date, but not necessarily an end date. If there is no end date, then the project is still in progress. There is a many to one relationship from project to customer. This relationship is represented by a foreign key in project pointing to a customer ID in customer table.

The employee relation has an ID and a password which they will use to login to the website. The Team relation has only two attributes, ID and name. There is a many to many relationship between these two relations because an employee can belong to many teams, and a team can have many employees. We also put salary on the many to many relation, we chose to do this because this way an employee can have a different salary depending on what team he is in.

The supplier relation and the materials relation have a many to many relationship because a supplier can sell multiple materials, and a certain material can be offered by many different suppliers. We decided to put cost and delivery time on the many to many relationship because depending on the supplier and material, the cost and delivery time would vary. It is also important to note that delivery time is determined by the number of days it takes to get from the supplier's warehouse to the construction site.

The phase relation holds the phases of a project. A project can have many phases, but one phase belongs to one project. The many to one relationship is represented by a foreign key referencing a project id. Every phase has a start date, but not necessarily an end date. If no end date is present, that phase is currently in progress. We are allowing only one phase per project to be currently in progress. It is achieved with the help of a trigger, more details in can be found in the triggers section. In addition, there's a many to many relation from phase to material. It allows to associate required materials for a phase and also holds the required material quantity attribute.

The task relation holds the tasks of a phase. This relation has a many to one relationship with phase, from task to phase. The relationship is represented by a foreign key on phase id. The cost, start date and end date attribute can vary in terms of value depending on which phase the task is referencing. If the end date is not present, then that task has not been completed yet.

The transaction salary relation keeps track of the payments made for an employee's salary, in regards to a specific project. This relation has a many to one relationship with the project, employee and team relations. By doing this we can easily look up which employee, in what team the employee was and on which project he was working in order to receive his salary.

The transaction material relation keeps track of the payments made for a particular material in a project. This relation has a many to one relationship with supplier, material and project. This was done so we can quickly look up which material, from a particular supplier, was paid for a particular project.

The transaction task relation is used to create a log for all the tasks that have been paid for a particular project. This relation has a many to one relationship with

task and project. Similarly like the other transaction tables, we did this so we can quickly look up what task has been paid.

Upon a new transaction of any of three types, the corresponding project's remaining budget is checked before the transaction occurs in order to make sure that there's enough funds. It is achieved with the help of triggers. More details can be found at the triggers section

By doing a union from all three transaction tables and summing the cost from each transaction, we calculate the current total cost paid for the house. By adding to this amount the current remaining budget, we can find the total budget amount invested in the project.

# Table Creation Queries

```
CREATE TABLE CUSTOMER(
        custID int unsigned not null unique auto_increment,
        custName varchar(255) not null,
        custPwd varchar(255) not null,
        custAddress varchar(255),
        custPhoneNum varchar(255),
        primary key (custID)
);

CREATE TABLE EMPLOYEE(
        empID int unsigned not null unique auto_increment,
        empName varchar(255) not null,
        empPwd varchar(255) not null,
        empAddress varchar(255),
        empPhoneNum varchar(255),
        primary key(empID)
);

CREATE TABLE TEAM(
        teamID int unsigned not null unique auto_increment,
        teamName varchar(255) not null,
        primary key (teamID)
);

CREATE TABLE TEAM_EMPLOYEE(
        empID int unsigned not null,
        teamID int unsigned not null,
        salary int unsigned not null,
        primary key (empID, teamID),
        foreign key (empID) references EMPLOYEE (empID),
        foreign key (teamID) references TEAM (teamID)
);

CREATE TABLE PROJECT(
    projID int unsigned not null unique auto_increment,
    custID int unsigned not null,
    projName varchar(255) not null,
    projDetails varchar(255),
    budgetAmount float(12,2) unsigned,
    estimatedCost float(12,2) unsigned,
    projDateStart date,
    projDateEnd date,
    primary key (projID),
    foreign key (custID) references CUSTOMER (custID)
);

ALTER TABLE PROJECT ALTER budgetAmount SET DEFAULT 0;
```

```
CREATE TABLE PROJECT_TEAM(
        projID int unsigned not null,
        teamID int unsigned not null,
        primary key (projID, teamID),
        foreign key (projID) references PROJECT (projID),
        foreign key (teamID) references TEAM (teamID)
);

CREATE TABLE PHASE(
        phaseID int unsigned not null unique auto_increment,
        projID int unsigned not null,
        phaseName varchar(255) not null,
        phaseDetails varchar(255),
        phaseDateStart date,
        phaseDateEnd date,
        primary key (phaseID),
        foreign key (projID) references PROJECT (projID)
);

CREATE TABLE TASK(
        taskID int unsigned not null unique auto_increment,
        phaseID int unsigned not null,
        taskDetails varchar(255) not null,
        taskCost float(12,2) unsigned,
        taskEstimateHours int unsigned not null,
        taskDateStart date,
        taskDateEnd date,
        primary key (taskID),
        foreign key (phaseID) references PHASE(phaseID)
);

CREATE TABLE MATERIAL(
        matID int unsigned not null unique auto_increment,
        matName varchar(255) not null,
        primary key (matID)
);

CREATE TABLE PHASE_MATERIAL(
        phaseID int unsigned not null,
        matID int unsigned not null,
        qty int unsigned not null,
        primary key (phaseID, matID),
        foreign key (phaseID) references PHASE (phaseID),
        foreign key (matID) references MATERIAL (matID)
);

ALTER TABLE PHASE_MATERIAL ALTER qty SET DEFAULT 1;

CREATE TABLE SUPPLIER(
        supID int unsigned not null unique auto_increment,
```

```
        supName varchar(255) not null,
        supAddress varchar(255),
        supPhoneNum varchar(255),
        primary key (supID)
);

CREATE TABLE MATERIAL_SUPPLIER(
        matID int unsigned not null,
        supID int unsigned not null,
        matCost float(12,2) unsigned,
        deliveryTime int unsigned,
        primary key (matID, supID),
        foreign key (matID) references MATERIAL (matID),
        foreign key (supID) references SUPPLIER (supID)
);

ALTER TABLE MATERIAL_SUPPLIER ALTER matCost SET DEFAULT 0;

CREATE TABLE TRANSACTION_TASK(
   transID int unsigned not null unique auto_increment,
        projID int unsigned not null,
   taskID int unsigned not null,
   transCost float(12,2) unsigned not null,
   transDate date not null,
   primary key (transID),
        foreign key (projID) references PROJECT (projID),
   foreign key (taskID) references TASK (taskID)
);


CREATE TABLE TRANSACTION_SALARY(
   transID int unsigned not null unique auto_increment,
        projID int unsigned not null,
   empID int unsigned not null,
   teamID int unsigned not null,
   transCost float(12,2) unsigned not null,
   transDate date not null,
   primary key (transID),
        foreign key (projID) references PROJECT (projID),
   foreign key (empID) references EMPLOYEE (empID),
   foreign key (teamID) references TEAM (teamID)
);

CREATE TABLE TRANSACTION_MATERIAL(
   transID int unsigned not null unique auto_increment,
        projID int unsigned not null,
   matID int unsigned not null,
   supID int unsigned not null,
   transCost float(12,2) unsigned not null,
   transDate date not null,
   primary key (transID),
```

```
        foreign key (projID) references PROJECT (projID),
    foreign key (matID) references MATERIAL (matID),
    foreign key (supID) references SUPPLIER (supID)
);
```

# Triggers

We made transaction triggers on our three transaction tables: transaction_salary, transaction_material, transaction_task which on before insert check if the transaction cost would result in the project's remaining budget to go below 0$. If it is the case, an exception is thrown otherwise we update the project's remaining budget by subtracting the transaction cost from it.

```
delimiter $$
create trigger trans_task_trigger before insert on TRANSACTION_TASK
for each row
begin
declare currBudget float(12,2);
set currBudget = (select budgetAmount from PROJECT where projID = new.projID);
if ( currBudget - new.transCost ) < 0 then
 signal sqlstate '45000'
  set message_text = 'Task Transaction Exception! Not Enough Funds in the Project\'s Remaining
Budget';
else
 update PROJECT set budgetAmount = (currBudget - new.transCost) where
PROJECT.projID=new.projID;
end if;
end;$$


create trigger trans_material_trigger before insert on TRANSACTION_MATERIAL
for each row
begin
declare currBudget float(12,2);
set currBudget = (select budgetAmount from PROJECT where projID = new.projID);
if ( currBudget - new.transCost ) < 0 then
 signal sqlstate '45000'
  set message_text = 'Material Transaction Exception! Not Enough Funds in the Project\'s Remaining
Budget';
else
 update PROJECT set budgetAmount = (currBudget - new.transCost) where
PROJECT.projID=new.projID;
end if;
end;$$


create trigger trans_salary_trigger before insert on TRANSACTION_SALARY
for each row
begin
declare currBudget float(12,2);
set currBudget = (select budgetAmount from PROJECT where projID = new.projID);
```

```
if ( currBudget - new.transCost ) < 0 then
 signal sqlstate '45000'
  set message_text = 'Salary Transaction Exception! Material Exception Exception! Not Enough Funds
in the Project\'s Remaining Budget';
else
 update PROJECT set budgetAmount = (currBudget - new.transCost) where
PROJECT.projID=new.projID;
end if;
end;$$
```

Furthermore, we implemented triggers which prevent the update of transactions in all of the three transaction tables.

```
create trigger trans_task_update_trigger before update on TRANSACTION_TASK
for each row
begin
 signal sqlstate '45000'
  set message_text = 'Task Transaction Exception! Transactions cannot be updated...';
end;$$

create trigger trans_material_update_trigger before update on TRANSACTION_MATERIAL
for each row
begin
 signal sqlstate '45000'
  set message_text = 'Material Transaction Exception! Transactions cannot be updated...';
end;$$

create trigger trans_salary_update_trigger before update on TRANSACTION_SALARY
for each row
begin
 signal sqlstate '45000'
  set message_text = 'Salary Transaction Exception! Transactions cannot be updated...';
end;$$
```

More triggers were made to make sure that employee's and customer's password has at least eight characters in their password attribute.

```
create trigger cust_insert_trigger before insert on CUSTOMER
for each row
begin
 declare pwdLength int;
 set pwdLength = (select length(new.custPwd));
 if (pwdLength < 8) then
  signal sqlstate '45000'
        set message_text= 'Customer Password Exception! Password length must be at least 8
characters long...';
```

```
end if;
end;$$

create trigger cust_update_trigger before update on CUSTOMER
for each row
begin
 declare pwdLength int;
 set pwdLength = (select length(new.custPwd));
 if (pwdLength < 8) then
  signal sqlstate '45000'
        set message_text= 'Customer Password Exception! Password length must be at least 8
characters long...';
end if;
end;$$

create trigger emp_insert_trigger before insert on EMPLOYEE
for each row
begin
 declare pwdLength int;
 set pwdLength = (select length(new.empPwd));
 if (pwdLength < 8) then
  signal sqlstate '45000'
        set message_text= 'Employee Password Exception! Password length must be at least 8
characters long...';
end if;
end;$$


create trigger emp_update_trigger before update on EMPLOYEE
for each row
begin
 declare pwdLength int;
 set pwdLength = (select length(new.empPwd));
 if (pwdLength < 8) then
  signal sqlstate '45000'
        set message_text= 'Employee Password Exception! Password length must be at least 8
characters long...';
end if;
end;$$
```

We have also implemented triggers on the phase_material table to make sure that the quantity has at least a value of 1.

```
create trigger phase_mat_insert_trigger before insert on PHASE_MATERIAL
for each row
begin
 if (new.qty < 1) then
  signal sqlstate '45000'
        set message_text= 'Phase Material Exception! Quantity must be at least 1';
```

```
end if;
end;$$

create trigger phase_mat_update_trigger before update on PHASE_MATERIAL
for each row
begin
 if (new.qty < 1) then
  signal sqlstate '45000'
        set message_text= 'Phase Material Exception! Quantity must be at least 1';
end if;
end;$$
```

For project, task and phase tables, we have made triggers that test the start date and end date attributes. The behaviour is the following: if there's an end date but the start date is not set, then throw an exception; else test if the start date is less than the end date, if not the case than throw an exception. For phase table, we had to additionally check before insert/update that there's at most one phase per project which can be in progress (start date is set while end date is not set). If it's not the case, than throw an exception.

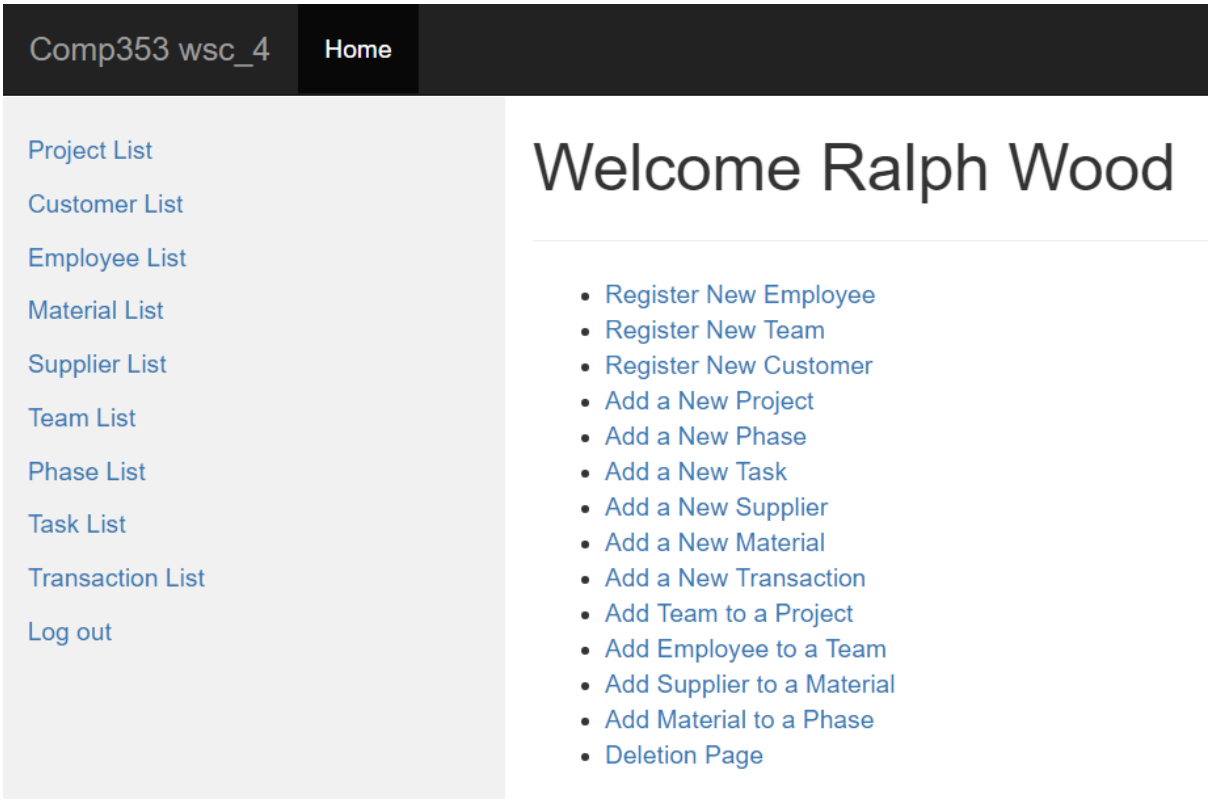
```
create trigger project_insert_trigger before insert on PROJECT
for each row
begin
 if (new.projDateEnd is not null) then
  if(new.projDateStart is null) then
   signal sqlstate '45000'
    set message_text = 'Project Exception! Cannot set an end date without a start date...';
  elseif (new.projDateStart > new.projDateEnd) then
   signal sqlstate '45000'
    set message_text = 'Project Exception! Start date is bigger than end date...';
  end if;
 end if;
end;$$

create trigger project_update_trigger before update on PROJECT
for each row
begin
 if (new.projDateEnd is not null) then
  if(new.projDateStart is null) then
   signal sqlstate '45000'
    set message_text = 'Project Exception! Cannot set an end date without a start date...';
  elseif (new.projDateStart > new.projDateEnd) then
   signal sqlstate '45000'
    set message_text = 'Project Exception! Start date is bigger than end date...';
  end if;
 end if;
end;$$
```

```
create trigger task_insert_trigger before insert on TASK
for each row
begin
 if (new.taskDateEnd is not null) then
  if(new.taskDateStart is null) then
   signal sqlstate '45000'
    set message_text = 'Task Exception! Cannot set an end date without a start date...';
  elseif (new.taskDateStart > new.taskDateEnd) then
   signal sqlstate '45000'
    set message_text = 'Task Exception! Start date is bigger than end date...';
  end if;
 end if;
end;$$

create trigger task_update_trigger before update on TASK
for each row
begin
 if (new.taskDateEnd is not null) then
  if(new.taskDateStart is null) then
   signal sqlstate '45000'
    set message_text = 'Task Exception! Cannot set an end date without a start date...';
  elseif (new.taskDateStart > new.taskDateEnd) then
   signal sqlstate '45000'
    set message_text = 'Task Exception! Start date is bigger than end date...';
  end if;
 end if;
end;$$


create trigger phase_insert_trigger before insert on PHASE
for each row
begin
declare numPhasesInProgress int;
 if (new.phaseDateEnd is not null) then
  if(new.phaseDateStart is null) then
   signal sqlstate '45000'
    set message_text = 'Phase Exception! Cannot set an end date without a start date...';
  elseif (new.phaseDateStart > new.phaseDateEnd) then
   signal sqlstate '45000'
    set message_text = 'Phase Exception! Start date is bigger than end date...';
  end if;
 else
  if(new.phaseDateStart is not null) then
   set numPhasesInProgress = (select count(*) from PHASE where phaseDateStart is not null and
phaseDateEnd is null and projID=new.projID);
   if(numPhasesInProgress > 0) then
          signal sqlstate '45000'
     set message_text = 'Phase Exception! There can be only one phase in progress at a time...';
   end if;
  end if;
```

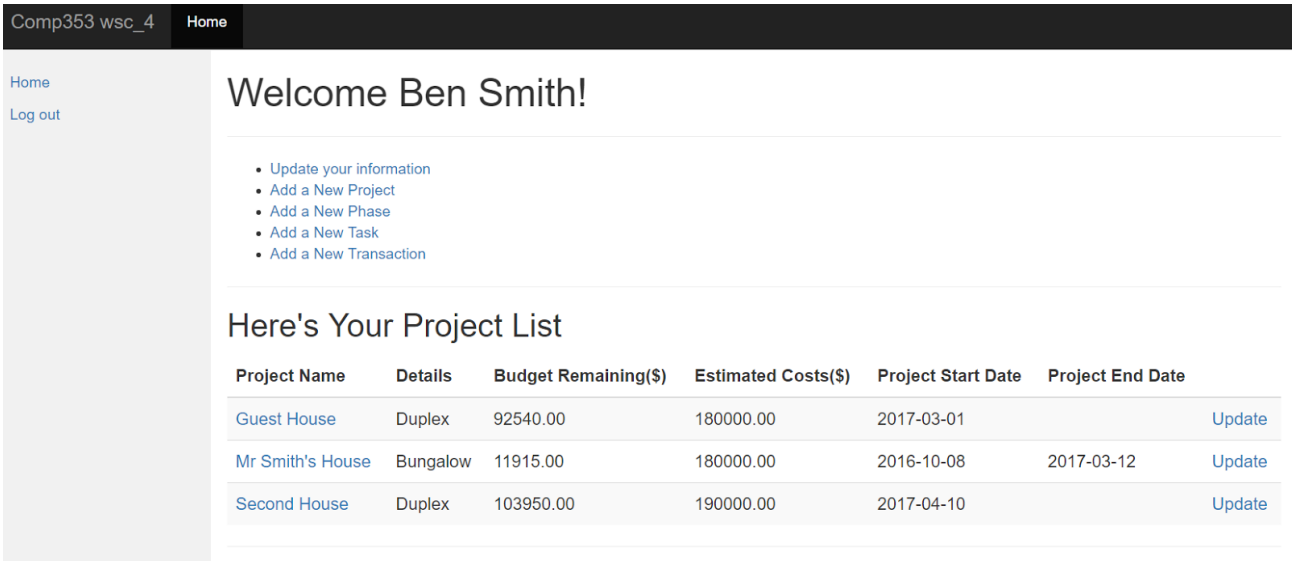
```
 end if;
end;$$

create trigger phase_update_trigger before update on PHASE
for each row
begin
declare numPhasesInProgress int;
 if (new.phaseDateEnd is not null) then
  if(new.phaseDateStart is null) then
   signal sqlstate '45000'
    set message_text = 'Phase Exception! Cannot set an end date without a start date...';
  elseif (new.phaseDateStart > new.phaseDateEnd) then
   signal sqlstate '45000'
    set message_text = 'Phase Exception! Start date is bigger than end date...';
  end if;
 else
  if(new.phaseDateStart is not null) then

   set numPhasesInProgress = (select count(*) from PHASE where phaseDateStart is not null and
phaseDateEnd is null and projID=new.projID);
   if(numPhasesInProgress > 0) then
           signal sqlstate '45000'
     set message_text = 'Phase Exception! There can be only one phase in progress at a time...';
   end if;
  end if;
 end if;
end;$$
```

# Web pages

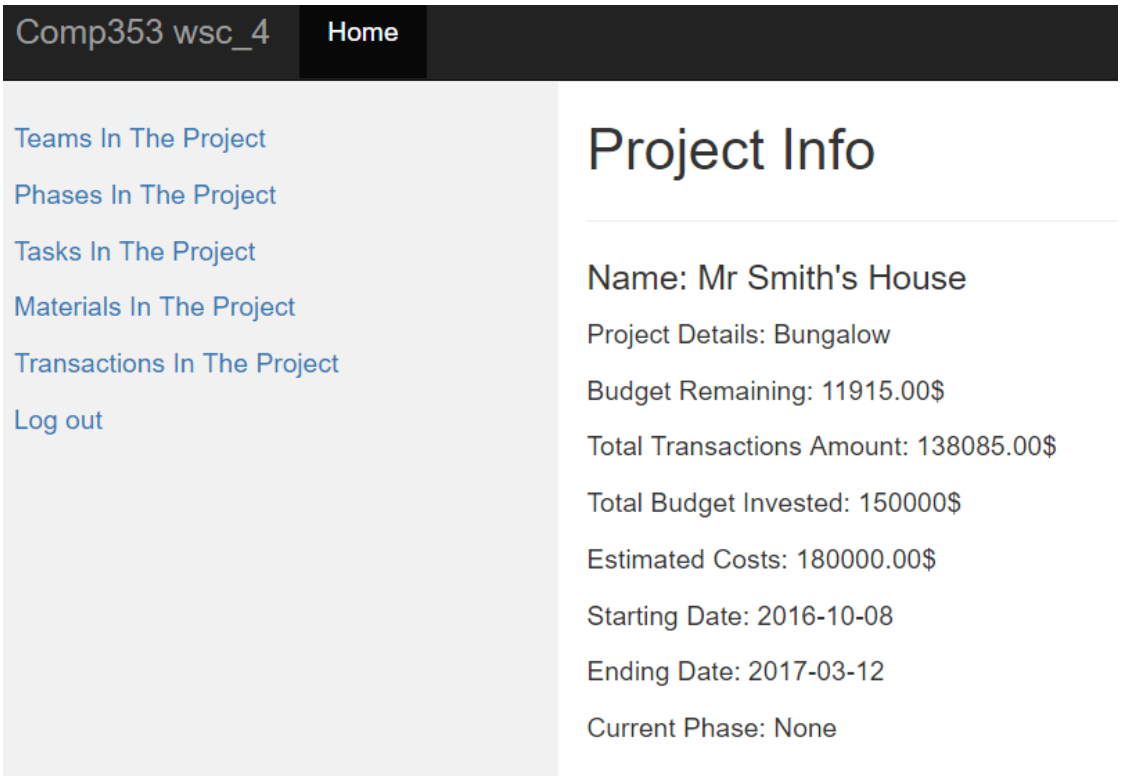
Our web page starts with the login page which is separated in two sections. One form is the login for employee while the other one is a login box for a customer. Depending on whether the customer or an employee had successfully logged in, a different page is shown. An employee is shown a page with links which allows him to add any data to any table in our database. It also provides him a link to a page allowing him to delete some values. We chose to allow the removal of just a couple of values in order to not break important dependencies. Affected tables are: team_employee, project_team, material_supplier and phase_material. An employee also has a side menu on the right which allows him to navigate different reports which output various contents of our database. In these reports (except transaction) he can edit any interested row by clicking the “update” hyperlink.




The customer on the other hand, has a limited access. On successful login, he is shown the list of his projects. He also has a limited set of actions to add new information to the database, which is limited to his projects.




Upon clicking on the project name's hyperlink, a new page with project information is brought up. The menu on the left allows to view various reports related to the project and also update them. An employee can get to the same project info page through project list page by clicking the project name hyperlink.




**proj_list.php:**

This page lists all the projects entered into the database. By clicking on a project name, it send you to the page proj_index.php, where more information about a project can be found. It also lists the project's budget remaining, estimated cost, start date, end date and project's customer name.

Queries used on this page:

```
SELECT PROJECT.projID, projName,  projDetails, budgetAmount, estimatedCost,
projDateStart, projDateEnd, custName
FROM PROJECT
JOIN CUSTOMER on CUSTOMER.custID=PROJECT.custID;
```

**proj_index.php:**

This page reveals important information in regards to a project such as: the budget remaining, total budget invested, the total transaction cost and the estimated cost.

Queries used on this page:

```
SELECT proj, sum(cost) as totCost FROM
(SELECT projID as proj, transCost AS cost FROM TRANSACTION_MATERIAL
UNION ALL
SELECT projID as proj,  transCost AS cost  FROM TRANSACTION_SALARY
UNION ALL
SELECT projID as proj, transCost as cost  FROM TRANSACTION_TASK) t1          WHERE
proj=$projID GROUP BY proj;
```

```
SELECT phaseID, phaseName, projName, phaseDetails, phaseDateStart, phaseDateEnd
FROM PHASE left join PROJECT on PHASE.projID=PROJECT.projID
WHERE PHASE.projID = $projID
AND phaseDateEnd is null
ORDER BY phaseDateStart;
```

**cust_list.php:**

This page shows information regarding a customer such as their name, address and phone number.

Queries used on this page:

*SELECT * FROM CUSTOMER;*

**emp_list.php:**

This page shows information regarding an employee such as their name, address and phone number.

Queries used on this page:

*SELECT * FROM EMPLOYEE;*

**mat_list.php:**

This page shows information regarding a material such as its name, cost and supplier.

Queries used on this page:

```
SELECT MATERIAL.matID, matName, MATERIAL_SUPPLIER.matCost, deliveryTime,
supName
FROM MATERIAL
LEFT JOIN MATERIAL_SUPPLIER on MATERIAL.matID=MATERIAL_SUPPLIER.matID
LEFT JOIN SUPPLIER on MATERIAL_SUPPLIER.supID=SUPPLIER.supID
ORDER BY matName,matCost;
```

**sup_list.php:**

This page shows useful information regarding a supplier such as their name and contact information.

Queries used on this page:

*SELECT * FROM SUPPLIER;*

**team_list.php:**

This page shows available teams and upon clicking a team name would go to a page which lists the team members information.

Queries used on this page:

*SELECT teamID, teamName from TEAM ORDER BY teamID;*

**phase_list.php:**

This page shows useful information regarding a phase such as its start date, end date, phase name and to which project it belongs to. Upon clicking the phase name, it would go to a page which lists the all the task details from that phase.

Queries used on this page:

*SELECT PHASE.phaseID, PHASE.projID, PROJECT.projName, phaseName, phaseDetails, phaseDateStart, phaseDateEnd*
*FROM PHASE LEFT JOIN PROJECT on PHASE.projID=PROJECT.projID*
*ORDER BY PHASE.projID, PHASE.phaseID;*

**task_list.php:**

This page shows useful information regarding a task such as its details, start date, end date, cost, estimated hours and to which phase it belongs too.

Queries used on this page:

*SELECT TASK.phaseID, TASK.taskID, PHASE.phaseName, taskDetails, taskCost, taskEstimateHours, taskDateStart, taskDateEnd*
*FROM TASK*
*LEFT JOIN PHASE on TASK.phaseID=PHASE.phaseID*
*ORDER BY PHASE.phaseID, TASK.taskID;*

**trans_list.php:**

This page shows useful information regarding transactions. Its separated into three categories: materials, salary and task. By clicking on the project name, it will go to a page showing project info and by clicking on a phase name, it will go to a page showing a list of all tasks in that phase.

Queries used on this page:
*SELECT transID, PROJECT.projID, PROJECT.projName, MATERIAL.matID, matName, SUPPLIER.supID, supName, transCost, transDate*

```
FROM TRANSACTION_MATERIAL
JOIN PROJECT ON TRANSACTION_MATERIAL.projID=PROJECT.projID
JOIN MATERIAL ON TRANSACTION_MATERIAL.matID=MATERIAL.matID
JOIN SUPPLIER ON TRANSACTION_MATERIAL.supID=SUPPLIER.supID
ORDER BYtransDate";
```

```
SELECT transID, PROJECT.projID, PROJECT.projName, TASK.taskID, taskDetails,
transCost, transDate
FROM TRANSACTION_TASK
JOIN PROJECT on TRANSACTION_TASK.projID=PROJECT.projID
JOIN TASK on TRANSACTION_TASK.taskID=TASK.taskID
ORDER BY transDate;
```

```
SELECT transID, PROJECT.projID, PROJECT.projName, EMPLOYEE.empID, empName,
TEAM.teamID, teamName, transCost, transDate
FROM TRANSACTION_SALARY
JOIN PROJECT on TRANSACTION_SALARY.projID=PROJECT.projID
JOIN EMPLOYEE on TRANSACTION_SALARY.empID=EMPLOYEE.empID
JOIN TEAM on TRANSACTION_SALARY.teamID=TEAM.teamID
ORDER BY transDate;
```

**add_*.php:**

We have web pages that start with the prefix add_, which are used to insert new data into the database, those pages used the following queries:

```
INSERT INTO TRANSACTION_SALARY(projID,empID,teamID, transCost,transDate)
VALUE ($projID, $empID, $teamID, $cost, $transdate)";
```

```
INSERT INTO TRANSACTION_MATERIAL(projID,matID,supID, transCost,transDate)
VALUE ($projID, $matID, $supID, $cost, $transdate);
```

```
INSERT INTO TRANSACTION_TASK(projID,taskID,transCost,transDate) VALUE ($projID,
$taskID, $cost, $transdate);
```

```
INSERT INTO PROJECT_TEAM VALUES ($projID,$newID);
```

```
INSERT INTO TEAM(teamname) VALUES ('$name');
```

```
INSERT INTO
TASK(phaseid,taskdetails,taskcost,taskestimatehours,taskdatestart,taskdateend) VALUES
($phaseid,'$details',$cost,$estimate,$startdate,$enddate);
```

*INSERT INTO MATERIAL_SUPPLIER VALUES($matID,$supID,$cost,$time);*

*INSERT INTO SUPPLIER(supName,supAddress,supPhoneNum) VALUES*
*('$name','$addr','$phone');*

*INSERT INTO*
*PROJECT(custID,projName,projDetails,budgetAmount,estimatedCos,projDateStart,projDateEnd) VALUES*
*('$custID','$name','$details','$budget','$estimate',$startdat,$enddate);*

*INSERT INTO PHASE(projID,phaseName,phaseDetails,phaseDateStart,*
*phaseDateEd) VALUES*
*($projid,'$name','$details',$startdate,$enddate);*

*INSERT INTO PHASE_MATERIAL VALUES($phaseID,$matID,$qty);*

*INSERT INTO MATERIAL(matName) VALUES ('$matname');*

*INSERT INTO PHASE_MATERIAL VALUES ($phaseid,$newID,$qty);*

*INSERT INTO MATERIAL_SUPPLIER VALUES ($newID,$supid,$cost,$time);*

**remove_many_many.php:**

This page allows the user to remove any employee from a team, a team from a project, a material by a supplier and a material from a phase.

Queries used on this page:

*DELETE FROM TEAM_EMPLOYEE WHERE empID=$empID AND teamID=$teamID;*
*DELETE FROM PROJECT_TEAM WHERE projID=$projID AND teamID=$teamID;*
*DELETE FROM MATERIAL_SUPPLIER WHERE supID=$supID AND matID=$matID;*
*DELETE FROM PHASE_MATERIAL WHERE phaseID=$phaseID AND matID=$matID;*

**cust_index.php:**

This page shows a list of available projects associated to the customer that is logged in.

Queries used on this page:

*SELECT projID, projName, projDetails, budgetAmount, estimatedCost, projDateStart, projDateEnd*
*FROM PROJECT*
*WHERE PROJECT.custID=$custID*
*ORDER BY projName;*

**update_*.php:**

We have several pages with the prefix update_, which allows the user to update their current information regarding a project. The following queries were used:

*UPDATE CUSTOMER SET custName='$new_name', custPwd='$new_pwd', custAddress='$new_addr', custPhoneNum='$new_num' WHERE custID=$custID;*

*UPDATE EMPLOYEE SET empName='$new_name', empPwd='$new_pwd', empAddress='$new_addr', empPhoneNum='$new_num' WHERE empID=$empID;*

*UPDATE MATERIAL SET matName = '$new_name' WHERE matID = $matID;*

*UPDATE PHASE SET projID='$new_phase', phaseName='$new_name', phaseDetails='$new_dtl', phaseDateStart='$new_str', phaseDateEnd='$new_end' WHERE phaseID=$phaseID;*

*UPDATE PROJECT SET custID='$new_cust', projName = \"$new_name\", projDetails= '$new_dtl',budgetAmount = $new_bgt, estimatedCost = $new_est, projDateStart = $new_str, projDateEnd = $new_end WHERE projID = $projID;*

*UPDATE SUPPLIER SET supName='$new_name', supAddress='$new_addr', supPhoneNum='$new_num' WHERE supID=$supID;*