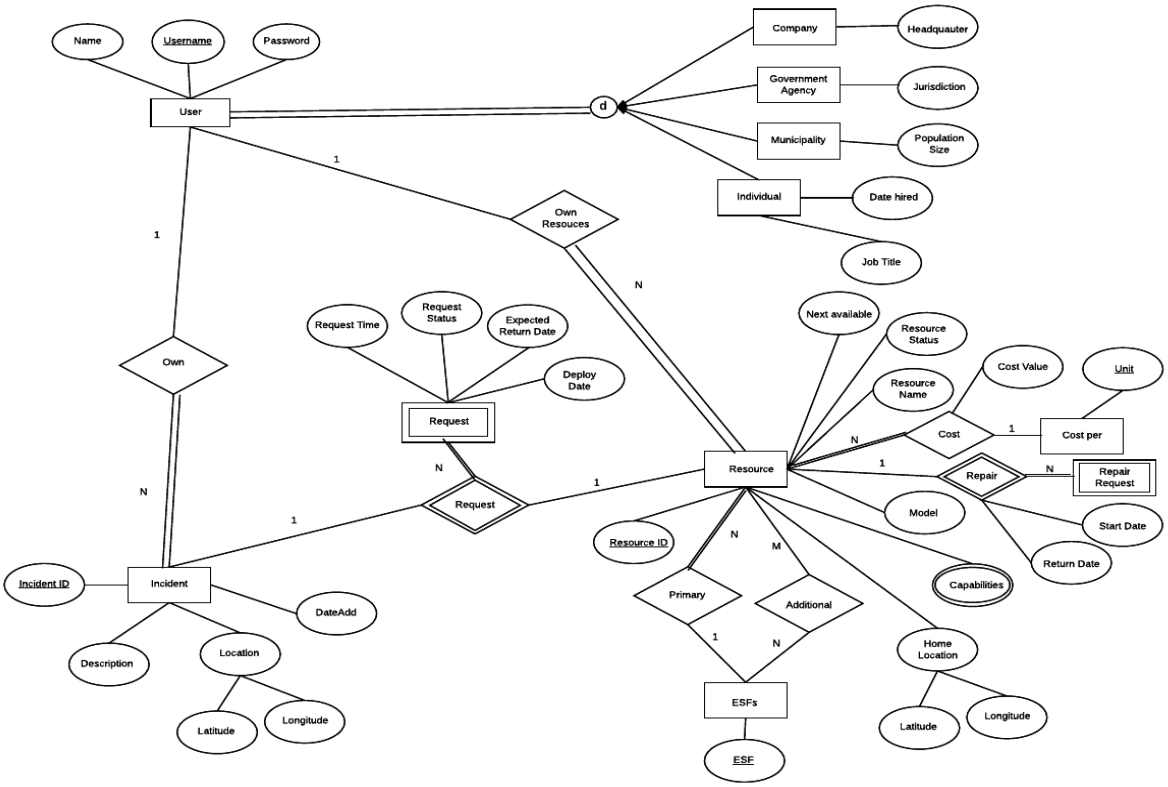
Project Phase 2

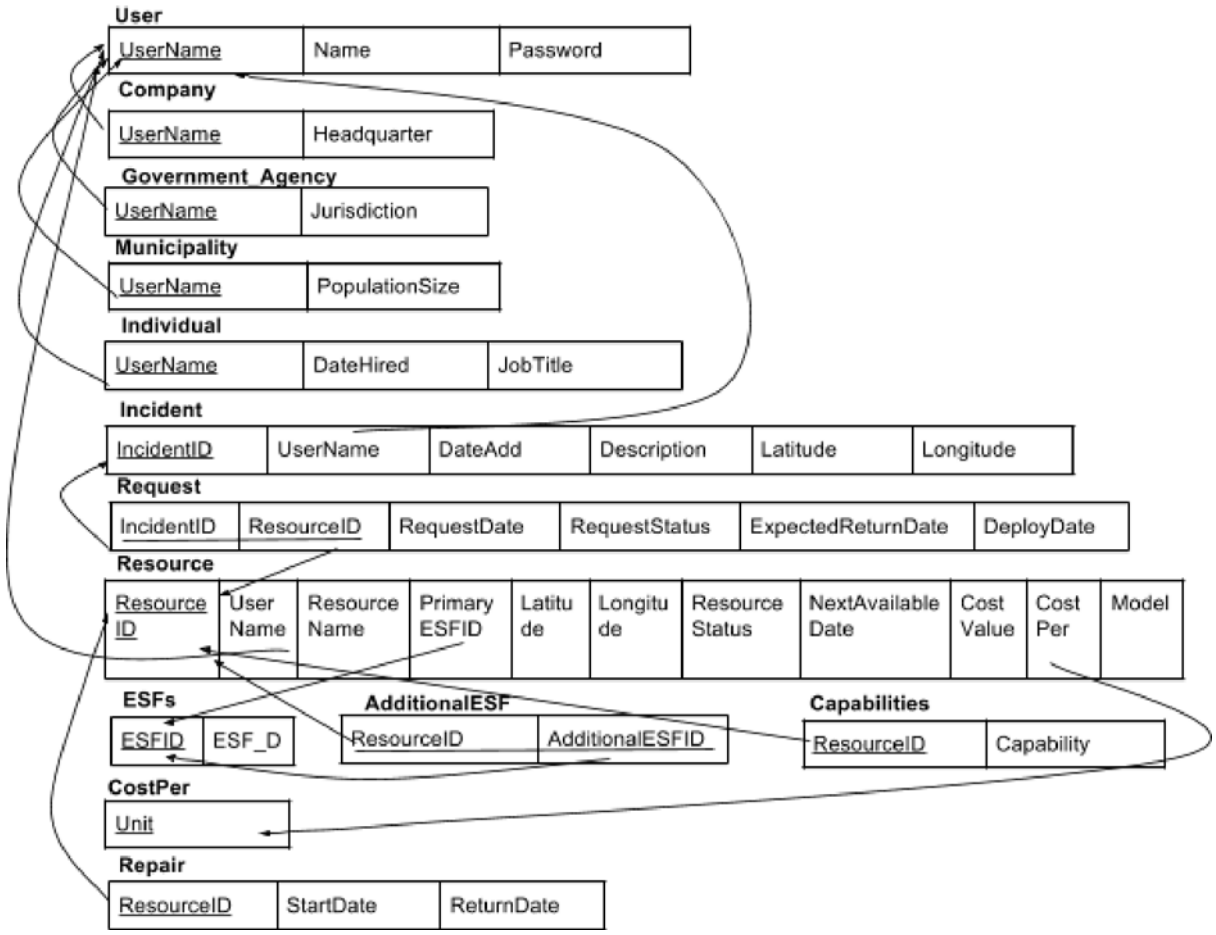
1. EER diagram

Here are the updates on EER diagram revised from Phase 1

1) add “repair request” entity with “startDate” and “returnDate” to store repair requests in database.
2) add “Cost Per” entity so that cost per entity would be flexible.
3) Add “ESFs” entity to handle primary and additional ESFs for resource
4) Add “DeployDate” in Request entity to record when the request been deployed.
5) change capabilities to multivalue attribute.


Name
Username
Password
User
d
Company
Headquauter
Government Agency
Jurisdiction
Municipality
Population Size
Individual
Date hired
Job Title
1
1
Own Resouces
N
Own
N
1
Request Time
Request Status
Expected Return Date
Deploy Date
Request
N
Request
1
1
Resource
Next available
Resource Status
Resource Name
Cost Value
Unit
Cost
1
Cost per
N
1
Repair
N
Repair Request
Model
Start Date
Return Date
Capabilities
Resource ID
N
M
Primary
Additional
1
N
ESFs
ESF
Home Location
Latitude
Longitude
Incident ID
Incident
DateAdd
Description
Location
Latitude
Longitude

2. EER to Relational mapping



3. SQL creat table statements to create schema

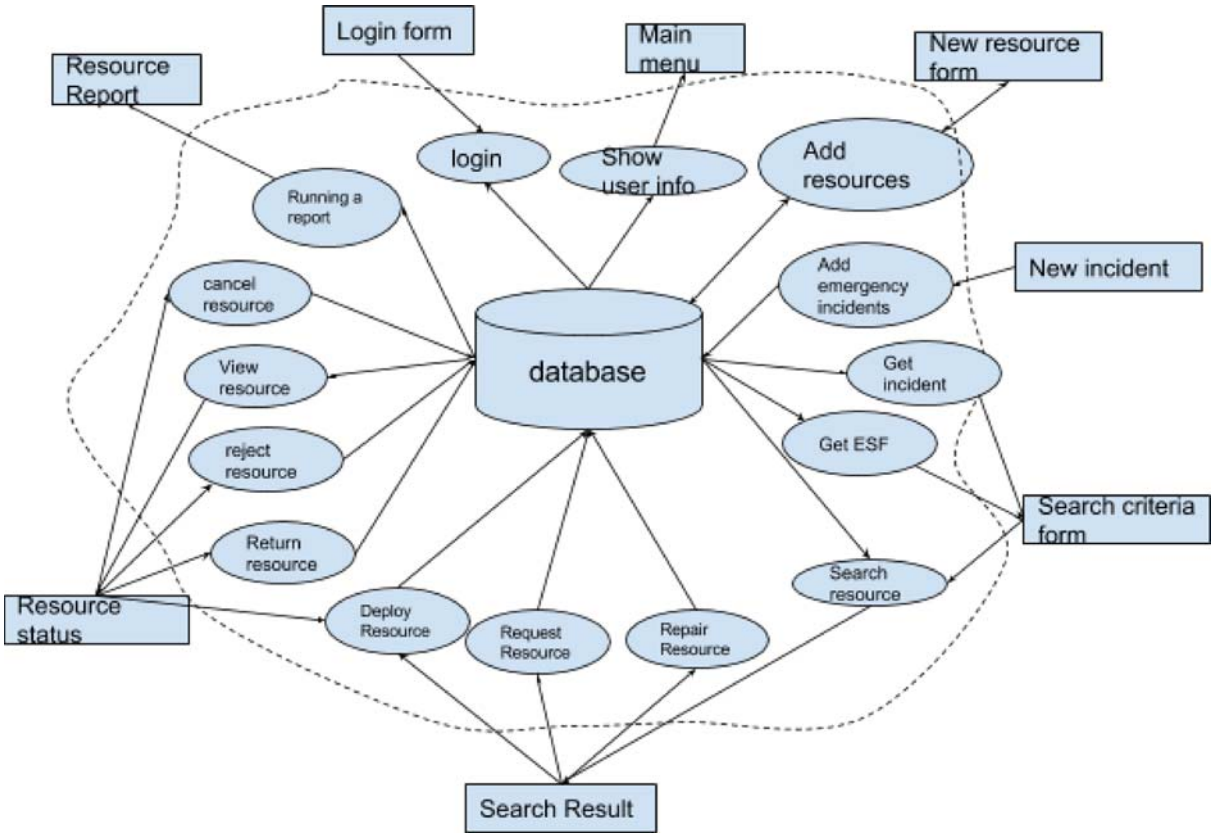
```
CREATE DATABASE IF NOT EXISTS erms_team26;
USE erms_team26;

DROP TABLE IF EXISTS `User`;
CREATE TABLE IF NOT EXISTS `User`(
 UserName VARCHAR(50) NOT NULL,
 `Name` VARCHAR(50) NOT NULL,
 `Password` VARCHAR(50) NOT NULL,
 PRIMARY KEY(UserName));

DROP TABLE IF EXISTS Company;
CREATE TABLE IF NOT EXISTS Company(
      UserName VARCHAR(50) NOT NULL,
      Headquarter VARCHAR(50) NOT NULL,
      PRIMARY KEY(UserName),
      FOREIGN KEY (UserName) REFERENCES `User`(UserName));

DROP TABLE IF EXISTS Government_Agency;
CREATE TABLE IF NOT EXISTS Government_Agency(
      UserName VARCHAR(50) NOT NULL,
      Jurisdiction VARCHAR(50) NOT NULL,
      PRIMARY KEY(UserName),
      FOREIGN KEY (UserName) REFERENCES `User`(UserName));

DROP TABLE IF EXISTS Municipality;
CREATE TABLE IF NOT EXISTS Municipality(
      UserName VARCHAR(50) NOT NULL,
      PopulationSize BIGINT NOT NULL,
      PRIMARY KEY(UserName),
      FOREIGN KEY (UserName) REFERENCES `User`(UserName));

DROP TABLE IF EXISTS Individual;
CREATE TABLE IF NOT EXISTS Individual(
      UserName VARCHAR(50) NOT NULL,
      DateHired DATE NOT NULL,
      Jobtitle VARCHAR(50) NOT NULL,
      PRIMARY KEY(UserName),
      FOREIGN KEY (UserName) REFERENCES `User`(UserName));

DROP TABLE IF EXISTS Incident;
CREATE TABLE IF NOT EXISTS Incident(
      IncidentID BIGINT NOT NULL AUTO_INCREMENT,
      Username VARCHAR(50) NOT NULL,
      DateAdd Date NOT NULL,
      Description VARCHAR(500) NOT NULL,
      Latitude FLOAT NOT NULL,
      Longitude FLOAT NOT NULL,
      PRIMARY KEY(IncidentID),
      FOREIGN KEY (UserName) REFERENCES `User`(UserName));


DROP TABLE IF EXISTS ESFs;
CREATE TABLE IF NOT EXISTS ESFs(
      ESFID INT NOT NULL,
      ESF_D VARCHAR(100) NOT NULL,
      PRIMARY KEY(ESFID));
```

```
DROP TABLE IF EXISTS CostPer;
CREATE TABLE IF NOT EXISTS CostPer(
      Unit VARCHAR(50) NOT NULL,
      PRIMARY KEY(Unit));

DROP TABLE IF EXISTS Resource;
CREATE TABLE IF NOT EXISTS Resource(
      ResourceID BIGINT NOT NULL AUTO_INCREMENT,
      UserName VARCHAR(50) NOT NULL,
      ResourceName VARCHAR(50) NOT NULL,
      Model VARCHAR(50) NOT NULL,
      costvalue FLOAT NOT NULL,
      cost_per VARCHAR(50) NOT NULL,
      Primary_ESFID INT NOT NULL,
      Latitude FLOAT NOT NULL,
      Longitude FLOAT NOT NULL,
      Resource_Status VARCHAR(50) NOT NULL,
      NextAvailableDate VARCHAR(50) NOT NULL,
      PRIMARY KEY(ResourceID),
      FOREIGN KEY (UserName) REFERENCES `User`(UserName),
      FOREIGN KEY (cost_per) REFERENCES CostPer(Unit),
      FOREIGN KEY (Primary_ESFID) REFERENCES ESFs(ESFID));

DROP TABLE IF EXISTS AdditionalESFs;
CREATE TABLE IF NOT EXISTS AdditionalESFs(
      ResourceID BIGINT NOT NULL,
      AdditionalESFID INT NOT NULL,
      UNIQUE(ResourceID,AdditionalESFID),
      FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID),
      FOREIGN KEY (AdditionalESFID) REFERENCES ESFs(ESFID));

DROP TABLE IF EXISTS Capabilities;
CREATE TABLE IF NOT EXISTS Capabilities(
      ResourceID BIGINT NOT NULL,
      Capability VARCHAR(100) NOT NULL,
      PRIMARY KEY(ResourceID),
      FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID));



DROP TABLE IF EXISTS Repair;
CREATE TABLE IF NOT EXISTS Repair(
      ResourceID BIGINT NOT NULL,
      StartDate DATE NOT NULL,
      ReturnDate DATE NOT NULL,
      PRIMARY KEY(ResourceID),
      FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID));







DROP TABLE IF EXISTS Request;
CREATE TABLE IF NOT EXISTS Request(
      ResourceID BIGINT NOT NULL,
```

```
    IncidentID BIGINT NOT NULL,
    RequestTime DATETIME NOT NULL,
    RequestStatus VARCHAR(50) NOT NULL,
    ExpectedReturnDate DATE NOT NULL,
    DeployDate DATE,
    UNIQUE(ResourceID,IncidentID),
    FOREIGN KEY (ResourceID) REFERENCES Resource(ResourceID),
    FOREIGN KEY (IncidentID) REFERENCES Incident(IncidentID));
```

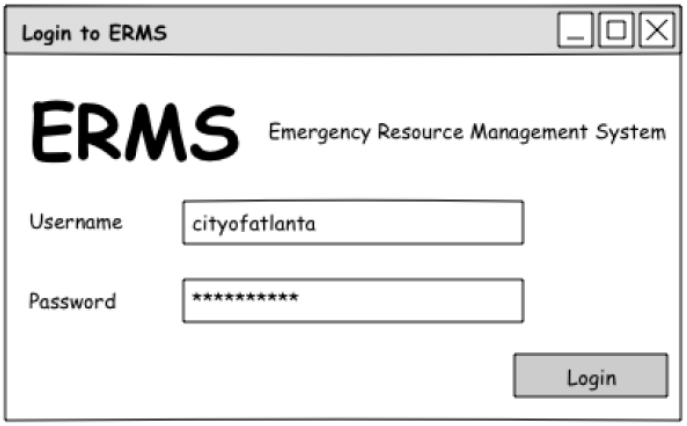
4. Abstruct Code with SQL

The task designs with the portion of the abstract code that refers to the EER diagram replaced with the corresponding SQL that refers the relations above. (50%)



The Following tasks are mainly based on the revised IFD diagram from project phase 1 It contains several main parts incluing:

- Login
- Main Menu
- Add New Resource
- Add New Incident
- Search Resources
- Search Results
    - Deploy resource
    - Repair resource
    - Request resource
- Resource Status
    - View resource
    - Return resource
    - Cancel resource requested by user
    - Deploy resource request received by user
    - Reject resource request received by user
    - Cancel repair
- Resource Report
    - Run resource report

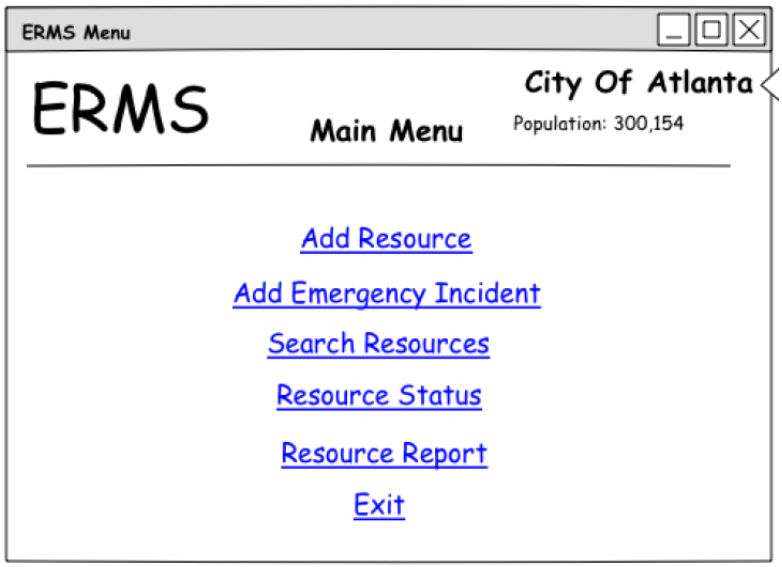


Main Task 1 - Login

// assume $UserName and $Password of current user is managed by application

```
SELECT * FROM User Where UserName='$UserName' and Password='$Password'
```

//If return none, application return error message “username and password invalid”

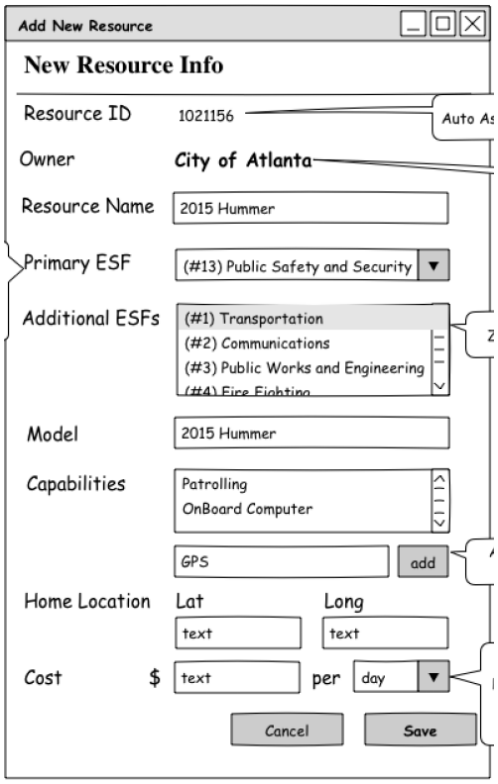


Main Task 2 - Main Menu

// assume $UserName of current user is managed by application
// assume the menu options are managed by the application
// display the user`s name plus

- the population size, if the user is a municipality,
- the jurisdiction, if the user is a government agency, or
- the location of the headquarters, if the user is a company.

```
SELECT User.name as Displayname,
       CASE WHEN C.UserName IS NOT NULL THEN CONCAT('Headquarter: ',C.Headquar
       WHEN G.UserName IS NOT NULL THEN CONCAT('Jurisdiction: ',G.Jurisdiction
       WHEN M.UserName IS NOT NULL THEN CONCAT('Population: ',I.PopulationSize
       WHEN I.UserName IS NOT NULL THEN '' END as Displayplus
FROM User
LEFT JOIN Company AS C
ON User.UserName=C.UserName
LEFT JOIN Government_Agency AS G
ON User.UserName=G.UserName
LEFT JOIN Municipality AS M
ON User.UserName= M.UserName
LEFT JOIN Individual AS I
ON User.UserName=I.UserName
WHERE User.UserName='$UserName'
```



Main Task 3 - Add New Resource

// populate dropdowns of Primary ESF, Additional ESFs and Cost per

```
SELECT CONCAT('(#',ESFID,')',ESF_D) AS displayESF
FROM ESFs;

SELECT Unit AS Displaycostper
FROM CostPer;
```

// read in parameters from form, parameter=read(“parameter name”)
//Validate form data

// assume $ResourceID, $UserName and $Name of current user, current date $Now, current datetime $Nowdatetime are managed by application, $ResourceStatus is “Available” by default.
// assume when insert new Resource the default NextAvailableDate is ‘Now’
//insert new Resource, Read values from the new Resource form $ResourceName, $Model, $costvalue, $cost_per, $Latitute, $Longtitude

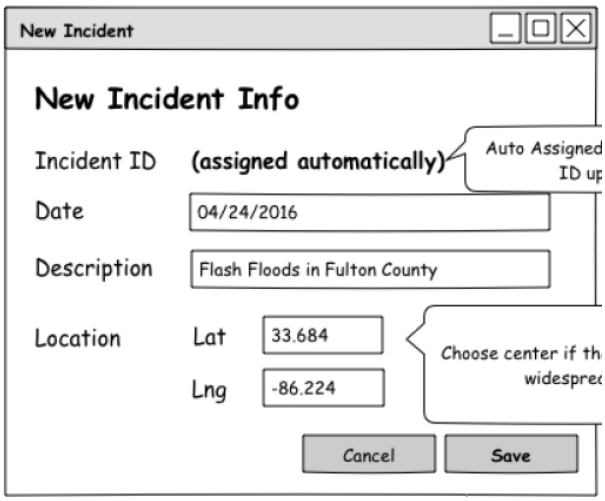
```
INSERT INTO Resource (ResourceID, UserName, ResourceName, Model,
costvalue, cost_per, Latitute, Longtitude, ResourceStatus,
NextAvailableDate) VALUES ($ResourceID, $UserName, $ResourceName,
$Model, $costvalue, $cost_per, $Latitute, $Longtitude,
'Available','Now')
```

//if there is Addtionall ESF added in new Resource then insert each $AddtionalESFID and its $ResourceID into AdditionalESFs table

```
INSERT INTO AdditionalESFs (ResourceID, AdditionalESFID) VALUES
($ResourceID, $AdditionalESFID)
```

//If there is capabilities added in new Resource then insert each $Capability and its $ResourceID into Capabilities table

```
INSERT INTO Capabilities (ResourceID, Capability) VALUES
($ResourceID, $Capability)
```

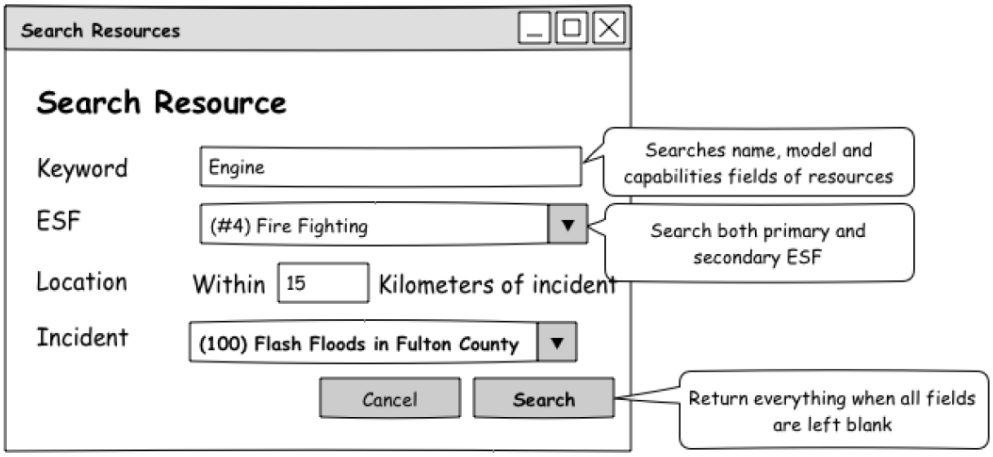


Main Task 4 - Add New Incident

//read in parametesr from form
// validate form data
// assume $IncidentID, $UserName is managed by application

//Insert new Incident, read from form $ResourceID, $UserName, $Date, $Description, $Latitute, $Longitude

```
INSERT INTO Incident (IncidentID, UserName, DateAdd, Description,
Latitute, Longitude) VALUES ($ResourceID, $UserName, $Date,
$Description, $Latitute, $Longitude)
```



Main Task 5 - Search Resources

// populate dropdowns, allowing to select blank option

```
SELECT CONCAT('(#',ESFID,')',ESF_D) AS displayESF
FROM ESFs;

SELECT CONCAT('(',IncidentID,')', Description)  FROM Incident;
```

//read in parameters from form, allowing read empty string
$Keyword=read(“Keyword”)
$ESFID=read(“ESF”) //if ESF is empty then $ESFID is NULL, otherwise it is an INT
$Location=read(“Location”)
$IncidentID=read(“Incident”)

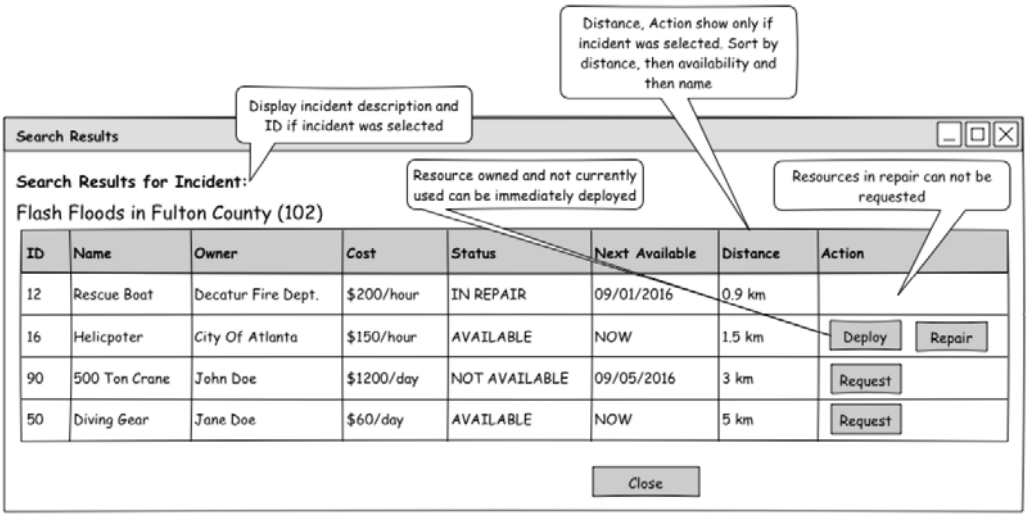
//validate form data, allowing empty strings
If($Location<0) then
    error (“Distance must be greater than Zero.”)
end if
//Show Search Result when there is no incident selected in search criteria form

```
SELECT R.ResourceID,
       R.ResourceName,
       U.Name as Onwer,
       CONCAT(R.costvalue, '/', R.cost_pert) as cost,
       ResourceStatus,
       NextAvailableDate
FROM Resource AS R
INNER JOIN User U
on U.UserName=R.UserName
LEFT JOIN Capabilities AS C
ON R.ResourceID=C.ResourceID
LEFT JOIN AdditionalESFs AS E
ON C.ResourceID=E.ResourceID
WHERE (ResourceName LIKE '%{$Keyword}%' or  Model LIKE
'%{$Keyword}%' or C.Capability LIKE '%{$Keyword}%' or
$Keyword='')
      and (Primary_ESFID =$ESFID or E.AdditionalESFID =$ESFID
or $ESFID is NULL)
GROUP BY 1,2,3,4,5,6;
```

// Show incident info and two additional columns if incident was selected

```
If ($IncidentID is not NULL) then
// Show Search Result Incident description
SELECT CONCAT(Description,'(', IncidentID,')') FROM Incident WHERE
IncidentID=$IncidentID

// dispaly search result
SELECT TEMP1.ResourceID,
       TEMP1.ResourceName,
       TEMP1.ResourceOwner,
       TEMP1.Cost,
       TEMP1.ResourceStatus,
       TEMP1.NextAvailableDate,
       CONCAT(TMEP1.Distance, ' km'),
       CASE WHEN TEMP1.ResourceStatus='In Repair' Then ''
            WHEN TEMP1.ResourceOwnerID!=TEMP1.IncidentOwnerID and
TEMP1.ResourceStatus!='In Repair' THEN 'Request'
            WHEN TEMP1.ResourceOwnerID=TEMP1.IncidentOwnerID and
TEMP1.ResourceStatus='Available' THEN 'Deploy, Repair'
            WHEN TEMP1.ResourceOwnerID=TEMP1.IncidentOwnerID and
TEMP1.ResourceStatus='In Use' THEN 'Repair'
       END AS Action
FROM(
    SELECT I.IncidentID,
           R.ResourceID,
           R.ResourceName,
           R.ResourceOwner,
```

```
            R.ResourceOwnerID,
            I.UserName as IncidentOwnerID,
            R.cost,
            R.ResourceStatus ,
            R.NextAvailableDate,
            @dLat:=RADIANS(I.Latitude-R.Latitude),
            @dLon:=RADIANS(I.Longitude-R.Longitude),
@a:=SIN(@dLat/2)*SIN(@dLat/2)+COS(I.Latitude)*COS(R.Latitude)*SIN(
@dLon/2)*SIN(@dLon/2),
            @c:=2*ATAN2(SQRT(@a),SQRT(1-@a) )*6371 as Distance
    FROM (SELECT IncidentID, UserName, Latitude, Longitude from
Incident WHERE IncidentID=$IncidentID) AS I
    CROSS JOIN (
          SELECT R1.ResourceID,
                 R1.ResourceName,
                 U.Name as ResourceOwner,
                 R1.UserName as ResourceOwnerID,
                 CONCAT(R1.costvalue, '/', R1.cost_pert) as cost,
                 R1.ResourceStatus,
                 R1.NextAvailableDate,
                 R1.Latitude,
                 R1.Longitude,
          FROM Resource AS R1
          INNER JOIN User U
          ON U.UserName=R1.UserName
          LEFT JOIN Capabilities AS C
          ON R.ResourceID=C.ResourceID
          LEFT JOIN AdditionalESFs AS E
          ON C.ResourceID=E.ResourceID
          WHERE (ResourceName LIKE '%{$Keyword}%' or  Model LIKE
'%{$Keyword}%' or C.Capability LIKE '%{$Keyword}%' or $Keyword='')
                and (Primary_ESFID =$ESFID or E.AdditionalESFID
=$ESFID or $ESFID is NULL)
          GROUP BY 1,2,3,4,5,6,7,8,9
     ) R
) TEMP1
WHERE TEMP1.Distance<=$Location
ORDER BY TEMP1.Distance, TEMP1.ResourceName ASC;
```



Main Task 6 - Search Results

Sub Task 1: Deploy resource

// application provides $ResourceID, $IncidentID, $Now , $N , $NowTime of current resource
// note $Now datatype as Date and $NowTime datatype as Datetime
// if user clicks “Deploy”

```
INSERT INTO Request (IncidentID, ResourceID, RequestTime, RequestStatus,
ExpectedReturnDate, DeployDate)
VALUES ($IncidentID, $ResourceID, $NowTime, “Accepted”, $Now+INTERVAL $N
DAY , $Now)
```

/* Here the ExpectedReturnDate is set to $Now+INTERVAL $N DAY since the we assume the user need to tell the system will use it N days when deploy his/her own resource */

//update ResourceStatus and NextAvailableDate in Resource Table

```
UPDATE Resource
SET ResourceStatus='InUse', NextAvailableDate=CAST($Now+INTERVAL $N DAY
AS VARCHAR)
WHERE ResourceID=$ResourceID
```

Sub Task 2: Repair resource

// application provides $ResourceID, $ResourceStatus, $NextAvailableDate
// if user clicks “Repair”, then user need to give the days N need to repair through application
// ($N is managed by application)
// insert new repair request

```
IF ($ResourceStatus='Availavle') THEN
   INSERT INTO Repair (ResourceID, StartDate, ReturnDate)
   VALUES ($ResourceID, $Now, $Now+$N)

   UPDATE Resource
   SET ResourceStatus='In Repair', NextAvailableDate=CAST($Now+INTERVAL
$N DAY AS VARCHAR)
   WHERE ResourceID=$ResourceID
END IF

IF ($ResourceStatus='In Use') THEN
INSERT INTO Repair (ResourceID, StartDate, ReturnDate)
            VALUES ($ResourceID,
str_to_date($NextAvailableDate,'%Y-%m-%d'),
str_to_date($NextAvailableDate,'%Y-%m-%d')+INTERVAL $N DAY)
END IF
```

Sub Task 3: Request resource

// if user clicks “Request”, an expected return date ($ReturnDate) must be given by the user
// through application

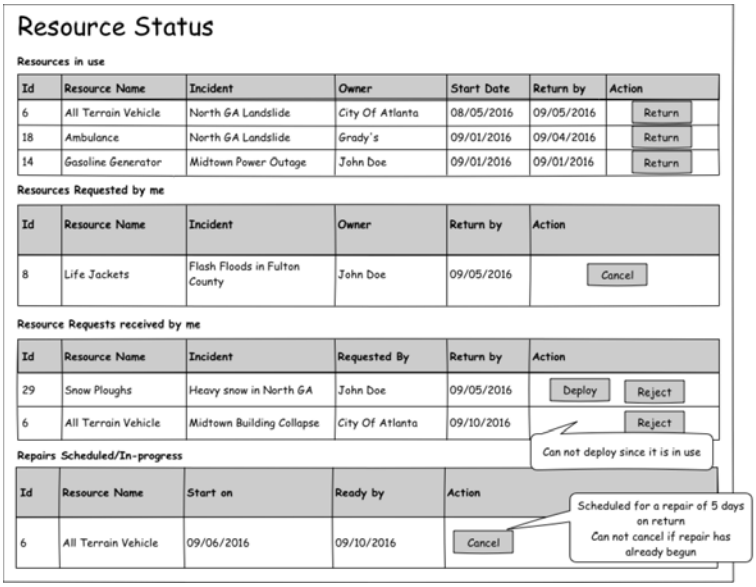
// read in parameters from form
 $RequestTime = $NowTime
 $ExpectedReturnDate = read(“ExpectedReturnDate”)

//validate form data
IF ($ExpectedReturnDate < Resource.NextAvailableDate) THEN
        error (“Expected return date must be bigger than next abvailable date.”))
        return to form
END IF

// insert new request

```
INSERT INTO Request (IncidentID, ResourceID, RequestTime, RequestStatus,
ExpectedReturnDate)
```

```
VALUES ($IncidentID, $ResourceID, $NowTime, 'Pending',
$ExpectedReturnDate);
```



Main Task 7 - Resource Status

Sub Task 1: View resource

// application provides $UserName of current user

// show resources in use

```
SELECT R.ResourceID, R.ResourceName, I.Description, U.Name,
RE.DeployDate, RE.ExpectedReturnDate,'Return' as Action
FROM Request AS RE
INNER JOIN Resource AS R ON RE.ResourceID = R.ResourceID
INNER JOIN Incident AS I ON RE.IncidentID = I.IncidentID
INNER JOIN User AS U on U.UserName=R.UserName
WHERE I.UserName=$UserName AND R.ResourceStatus='InUse' AND
RE.RequestStatus='Accepted';
```

// show resources requested by user

```
SELECT R.ResourceID, R.ResourceName, I.Description, I.IncidentID, U.Name
as Owner, RE.ExpectedReturnDate, 'Cancel' as Action
FROM Request AS RE
INNER JOIN Resource AS R ON RE.ResourceID = R.ResourceID
INNER JOIN Incident AS I ON RE.IncidentID = I.IncidentID
INNER JOIN User AS U on U.UserName=R.UserName
WHERE I.UserName=$UserName AND RE.RequestStatus='Pending';
```

// show resource requests received by user

```
SELECT R.ResourceID, R.ResourceName, I.Description, I.IncidentID, U.Name
as Request_by, RE.ExpectedReturnDate,
CASE WHEN R.ResourceStatus='Available' THEN 'Deploy, Reject' WHEN
R.ResourceStatus='In Use' then 'Reject' END as Action
FROM Request AS RE
INNER JOIN Resource AS R ON RE.ResourceID = R.ResourceID
INNER JOIN Incident AS I ON RE.IncidentID = I.IncidentID
INNER JOIN User AS U on U.UserName=I.UserName
WHERE R.UserName=$UserName AND RE.RequestStatus='Pending';
```

// show repairs scheduled/in-progress

```
SELECT R.ResourceID, R.ResourceName, REP.StartDate, REP.ReturnDate
FROM Repair AS REP
INNER JOIN Resource AS R ON RE.ResourceID=R.ResourceID
WHERE R.UserName=$UserName AND R.ResourceStatus='In Repair';
```

Sub Task 2: Return resource

// application provides $ResourceID and $ReturnDate (“Return by” column) of current resource
// if user clicks “Return” for “Resource in use”

```
IF (str_to_date(Resource.NextAvailableDate,'%Y-%m-%d') = ReturnDate) THEN
UPDATE Resource
SET ResourceStatus='Available', NextAvailableDate='Now'
WHERE ResourceID=$ResourceID
ELSE
UPDATE Resource
SET ResourceStatus='In Repair'
WHERE ResourceID=$ResourceID;
```

Sub Task 3: Cancel Resource Requested by user

// application provides $ResourceID and $IncidentID of current resource request
// if user clicks “cancel” for “Resources Requested by me”

```
DELETE FROM Request
WHERE ResourceID=$ResourceID AND IncidentID=$IncidentID;
```

// here note Request have two keys together as primary key

Sub Task 4: Deploy resource request eceived by user

// application provides $ResourceID, $IncidentID and $ExpectedReturnDate of current
// resource request
// if user clicks “Deply” for “Resource Requests received by me”

```
UPDATE Resource
SET ResourceStatus='InUse', NextAvailableDate=CAST($ExpectedReturnDate as
VARCHAR)
WHERE ResourceID=$ResourceID;

UPDATE Request
SET RequestStatus='Accepted', DeployDate=$Now
WHERE ResourceID=$ResourceID AND IncidentID=$IncidentID;
```

Sub Task 5: Reject resource request received by user

// application provides $ResourceID and $IncidentID of current resource
// if user clicks “Reject” for “Resource Requests received by me”

```
UPDATE Request
SET RequestStatus='Rejected'
WHERE ResourceID=$ResourceID AND IncidentID=$IncidentID;
```

Sub Task 6:: Cancel repair

// application provides $ResourceID of current resource

```
DELETE FROM Repair
WHERE ResourceID=$ResourceID
```

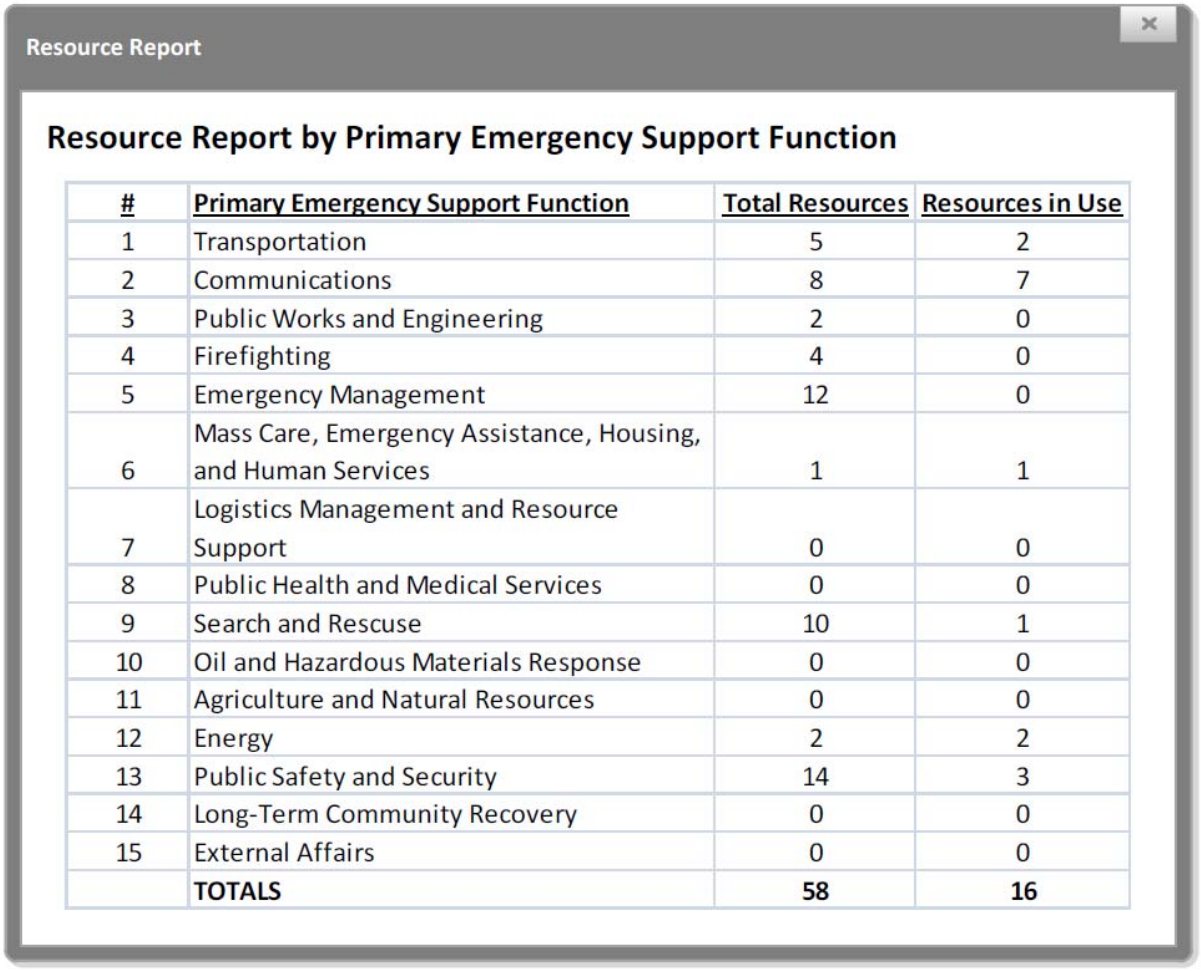


Figure 8 - Resource Report

Main Task 8 - Resource Report

// show statistics of primary ESF
// Total Resources
// application provides $UserName of current User

```
SELECT E.ESFID,
       E.ESF_D,
       COALESCE(C.Total_cnt,0) as total_cnt,
```

```
       COALESCE(C.in_use_cnt,0) as in_use_cnt
FROM ESFs AS E
LEFT JOIN (
       SELECT Primary_ESFID,
       COUNT (*) as Total_cnt,
       sum(case when ResourceStatus='InUse' then 1 else 0 end) as in_use_cnt
    FROM Resource
    WHERE UserName=$UserName
    GROUP BY Primary_ESFID
) C
on E.ESFID=C.Primary_ESFID
ORDER BY E.ESFID ASC;

//get the total number show in the last row of Resource Report

SELECT COUNT (*) as Total_cnt,
       sum(case when ResourceStatus='InUse' then 1 else 0 end) as in_use_cnt
FROM Resource
WHERE UserName=$UserName;
```