**CS 6400 Team 04**

*Bonifield, Carvallo, Du, Thai*

**Data Types:**

|  |  |  |
| --- | --- | --- |
| **User** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Name | <string> | Not Null |
| Password | <MD5 char> | Not Null |
| Username | <string> | Not Null |
| Type | List<string> | Not Null |

**\*Does this table need a type to relate it as a relationship to the ones below it? – THAI: I don’t think**

|  |  |  |
| --- | --- | --- |
| **Municipality** | | |
| **Attribute** | **Data Type** | **Nullable** |
| City | <string> |  |
| State | <string> |  |
| Country | <string> |  |
| County | <string> |  |

|  |  |  |
| --- | --- | --- |
| **Individual\_User** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Job\_Title | <string> |  |
| Date\_Hired | <dateTime> |  |

|  |  |  |
| --- | --- | --- |
| **Government\_Agency** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Local\_Office | <string> | NOT\_NULL |
| Agency\_Name | <string> | NOT\_NULL |

|  |  |  |
| --- | --- | --- |
| **Company** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Headquarters | <string> | NOT NULL |
| No\_Of\_Employees | <string> |  |

|  |  |  |
| --- | --- | --- |
| **Declaration** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Abbreviation | <string> | Not Null |
| Incident\_description | <string> | NULL |

|  |  |  |
| --- | --- | --- |
| **Requests** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Status | <string> | NOT NULL |
| Return\_by | <dateTime> | NULL |
| Date\_requested | <dateTime> | NOT\_NULL |
| Requested\_by | <ID> | NOT\_NULL |
| Resource\_ID | <ID> | NOT\_NULL |
| Incident\_ID | <ID> | NOT\_NULL |

* **Is this correct? I think we need to store IDs of resource and Incident so we can query it. Like the Friendship relationship in the sample project. It records owner\_email and friend\_email for the friendship weak entity.**

|  |  |  |
| --- | --- | --- |
| **Cost\_per** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Unit | <string> | Not Null |

|  |  |  |
| --- | --- | --- |
| **ESF** | | |
| **Attribute** | **Data Type** | **Nullable** |
| ESF\_ID | <number> | Not Null |
| ESF\_description | <string> | Not Null |

|  |  |  |
| --- | --- | --- |
| **Resource** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Resource\_ID | <string> | Not Null |
| R\_Owner | <string> | Not Null |
| Status | <string> |  |
| Cost\_amt | <long> |  |
| Latitude | <string> |  |
| Longtitude | <string> |  |
| Resource\_Name | <string> |  |
| Model | <string> |  |
| Max\_Distance | <float> |  |
| Capabilities | <string> |  |
| Primary\_ESF | <ID> |  |
| Additional\_ESF | <ID> |  |
| Standard\_cost | <float> |  |

|  |  |  |
| --- | --- | --- |
| **Incident** | | |
| **Attribute** | **Data Type** | **Nullable** |
| Incident\_ID | <string> | Not Null |
| I\_Owner | <string> | Not Null |
| Longitude | <long> |  |
| Latitude | <long> |  |
| Description | <string> |  |
| Date | <DateTime> |  |

**Business Logic Constraints:**

The following section lists business logic constraints that cannot be reflected in EER.

* Municipality must have municipality category
* Companies must have headquarters and number of employees
* Government agencies must have agency name and local office
* Individual users must have job title and hired date
* The resource’s owner is automatically set to the logged in user
* Coordinates of a location should be signed
* Unique ID of incident should combine the abbreviation of the incident type with an automatically generated number unique to that type
* If an incident is spread-out, user should choose the central location as the location of the incident
* All incidents are private to users who create them, and incidents cannot be shared
* When users search for resources, the matching function should return all resources that match the searching criteria, regardless of the owner of the incidents
* When using keyword as searching criteria, the function should match substring in the name, model, and capability field
* When using ESF as searching criteria, the function should match primary or additional ESF
* When using location as search criteria, the function should match resources with a home location within the given radius
* The search results should be ordered first by distance in ascending order, then alphabetically by resource name
* A resource can be requested directly from the search results only when the user selected the incident on the searching criteria form
* Distance and action showed only if incident was selected
* Both available and used resources can be requested
* Resources must be returned to the available status before it can be deployed again
* The system should prevent resources being deployed to the same incidents ~~immediately~~ after it is returned
* The system should allow user to
  + Request or cancel a request for resource
  + Deploy or reject a resource; deploy should be disabled for resources that are in use
* Users cannot request their own resources
* Request ID is a combination of IncidentID and ResourceID

**Task Decomposition & Abstract Code**

*Formatted in the following way:*

***Lock Types:*** Action on the XXXX table

***Number of Locks:*** Single, Many, Etc.

***Enabling Conditions:*** None, many etc.

***Frequency:*** X times a day

***Consistency:*** not critical, order is not critical, order is critical, etc.

***Subtasks:*** Mother Task is not required, mother task req

1. ***LOGIN***

***Task Decomposition***

***Lock Types:*** Read only on User table

***Number of Locks:*** Single

***Enabling Conditions:*** None

***Frequency:*** XXX

***Consistency:*** not critical, order is not critical

***Subtasks:*** Mother Task is not required

***Decomposition****:* Not Required

***Abstract Code***

* User enters *email(‘$Email’), password(‘$Password’)* input fields.
* If data validation is successful for both *username* and *password* input fields then:
  + When ***Enter*** button is clicked:
    - If User record is found but User.password != ‘$Password’
      * Go back to **Login** form with error message
    - Else:
      * Store Login information as session variable ‘$UserId’
      * Go to **Main Menu**  page.
* Else *email* and *password* input fields are invalid, display **Login** form with error message

1. ***Main Menu***

***Task Decomposition***

***Lock Types:*** Read only on User table

***Number of Locks:*** Single

***Enabling Conditions:*** Enabled by a user’s login

***Frequency:*** XXX

***Consistency:*** not critical, order is not critical

***Subtasks:*** Mother Task is not required

***Decomposition****:* Not Required

***Abstract Code***

* User lands on **Main Menu** page:
* Run **Get User** task; load information about the current user
  + Lookup user in User.type in User table
  + Display User.Name
  + Switch User.Type:
    - Municipality
      * Display municipality category
    - Individual
      * Display… what should it display if user? Njust name
    - Government
      * Display agency name
    - Company
      * Display location of headquarters and number of employees
* Display links to:
  + Add Resource Page
  + Add Incident
  + Search Resources
  + Resource Status
  + Resource Report

1. ***Add Resource***

***Task Decomposition***

***Lock Types:*** Read only on ESF table, Read on Cost\_Per table, Insert to Resources tabkle

***Number of Locks:*** Several schema constructs are needed

***Enabling Conditions:*** Enabled by a user login and add of resource

***Frequency:*** Once

***Consistency:*** not critical, order is not critical

***Subtasks:*** All tasks must be done in order. Mother task is req to coordinate substasks

***Decomposition****:* Required

***Abstract Code***

* User clicked on ***Add Resource*** button from **Main Menu**
* Get User.Name from session query or run **Get User** task
* Run the **Get ESF & Costs** task; query for information about available ESFs and COSTs,
  + From the ESF table lookup available ESFs
  + Populate ESF dropdown in **Add Resource Form**
* Run **Get Costs** task: query for information on available cost pers
  + From Cost\_Per table, lookup available cost per units.
  + Populate *cost per* dropdown in **Add Resource Form**
* Auto-Assign Resource ID
* Display ***Add Resource Form***
* User enters *Resource\_Name(‘$Resource\_Name’), Model(‘$Model’), Capabilities(‘[$Capabilities]), Latitude(‘$Home.Latitude’), Longitude(‘$Home.Longitude’), Max\_Distance(‘$Max Distance), Cost(‘$Cost),*  input fields.
* User selects *Primary ESF(‘$Primary\_ESF’), Secondary\_ESFs (‘[$Primary\_ESF’]), Per (‘$Cost\_Per)* from the respective dropdowns
* If data validation is successful for input fields then:
  + When ***Save*** button is clicked:
    - Store Resource Information as new entry to Resource table
    - Go to **Main Menu**  page.
* Else *input* fields are invalid, display **Add Resource** form with error message

1. ***Add Incident***

***Task Decomposition***

***Lock Types:*** Read only on Declared\_As table, Insert to Incident table

***Number of Locks:*** Several schema constructs are needed

***Enabling Conditions:*** Enabled by a user login and add new incident

***Frequency:*** Once

***Consistency:*** not critical, order is critical

***Subtasks:*** All tasks must be done in order. Mother task is req to coordinate substasks

***Decomposition****:* Required

***Abstract Code***

* User clicked on ***Add Incident*** button from **Main Menu**
* Get User.Name from session query.
* Run the **Get Incident Declaration** task; query for information about available incident types
  + From the Declaration table, lookup available incident declarations and descriptions
  + Populate *Declaration* dropdown on ***New Incident Form***
* Display ***Add Incident Form***
* User enters *Date(‘$Date),Description(‘$Description’), Latitude(‘$Home.Latitude’), Longitude(‘$Home.Longitude’),*  input fields.
* User selects *Declaration(‘$Incident\_Type* from the respective dropdowns
* If data validation is successful for input fields then:
  + When ***Save*** button is clicked:
    - Auto-Assign Incident ID based on declaration
    - Store Incident Information as new entry to Incident table
    - Go to **Main Menu**  page.
* Else *input* fields are invalid, display **Add Incident** form with error message

1. ***Search Resources***

***Task Decomposition***

***Lock Types:*** Read only on ESF table and Incident table

***Number of Locks:*** Several schema constructs are needed

***Enabling Conditions:*** Enabled by a user login and resource search

***Frequency:*** several times a day (~200 )

***Consistency:*** not critical

***Subtasks:*** Mother task is not required

***Decomposition****:* Not Required

***Abstract Code***

* User clicked on ***Search Resources*** button from **Main Menu**
* Get User.Name from session query.
* Run the **Get ESF** task; (defined above)
* Run the **Get Incidents** task
  + From Incidents table, select all available incidents
* Display *Resource Search Form* with ESF and Incident dropdowns populated
* User enters *keyword(‘$args),* and *Location(‘$MaxLoc),* inputs.
* User selects ESF and Incident from dropdown menus.
* If data validation is successful for input fields then:
  + When ***Search*** button is clicked:
    - Query database for requested parameters
    - Go to **Search Resource Results**  page/task.
* Else *input* fields are invalid, display **Search Resources** form with error message

1. ***Search Results <-needs a lot of work abc***

***Task Decomposition***

***Lock Types:*** Read only on Incident table, Read on Resources, Insert to Requests table

***Number of Locks:*** Several schema constructs are needed

***Enabling Conditions:*** Enabled by a user login and incident search

***Frequency:*** Once

***Consistency:*** critical. All incidents and statuses need to be shown correctle

***Subtasks:*** Mother task is req to coordinate substasks

***Decomposition****:* Required

***Abstract Code***

* User clicked on ***Search*** button from **Search Resources**
* Get Incident.Name from session query.
* Run the **Search Incident Resources** task; query for information about associated resources with a given incident.
  + From Resources select all resources that match the criteria specified in the **Search Resources** form.
* Display *Incident Search Form* with Resource.ID, Resource.Name, Resource.Owner, Resource.Cost, Resource.Status, Resource.Available for each resource dedicated to the task.
* If Resource.Owner== User.Name && Resource.Status ==”Available” then display **Deploy** button.
* Elsif Resource.Owner!= User.Name && Resource.Status ==”Available” display **Request**  Button
* Else
  + Do not display a button

1. ***Deploy Resource <-needs a lot of work abc***

***Task Decomposition***

***Lock Types:*** Update to Requests table, Update to Resources Table

***Number of Locks:*** 2 db locks

***Enabling Conditions:*** Enabled by a user login and clicking **Deploy** button

***Frequency:*** Several times a day

***Consistency***: DB needs to be updated correctly. Critical

***Subtasks:*** None

***Decomposition****: Not* Required

***Abstract Code***

* User clicked on ***Deploy*** button from either **Search Resources**
* Get Incident.Name from session query.
* Run the **Update Resource Availability Task** task; update resource information
  + From Resources select resources that that was selected.
  + If Resource.Owner == CurrentUser
    - Update Resource.Status to “Not Available”
  + Else reject DB update
* Run the **Update Request Task** task; update request information
  + From Request table select Request where Request.Resource AND Request.Incident== CurrentResource AND Current Incident
  + If Request.Resource.Owner=CurrentUser:
    - Update Request.Status to “DEPLOYED”
  + Else
    - Reject DB Update.

1. ***Request Resource <-needs a lot of work abc***

***Task Decomposition***

***Lock Types:*** Insert to Resources Table

***Number of Locks:*** 2 db locks

***Enabling Conditions:*** Enabled by a user login and clicking **Request** button

***Frequency:*** Several times a day

***Consistency***: DB needs to be updated correctly. Critical

***Subtasks:*** None

***Decomposition****: Not* Required

***Abstract Code***

* User clicked on ***Request*** button from either **Search Resource Results**
* Get Incident.ID and Resouce.ID and User.ID from session query.
* Run the **Create Request Task** task; create new resource request
  + From Request table insert new request where Request.ID is a composed of Incident.ID and Resource.ID. Request.Requested\_by=CurrentUser, Date\_requested=Timestamp.Now, Request.Status = “REQ”
  + If Request.Resource.Owner=CurrentUser:
    - Update Request.Status to “DEPLOYED”
    - Run the **Update Resource Availability Task**
  + Else
    - Reject DB Update.

1. ***Cancel Request <-needs a lot of work abc***

***Task Decomposition***

***Lock Types:*** Update to Requests Table

***Number of Locks:*** 2 db locks

***Enabling Conditions:*** Enabled by a user login and clicking **Cancel** button

***Frequency:*** Several times a day

***Consistency***: DB needs to be updated correctly. Critical

***Subtasks:*** None

***Decomposition****: Not* Required

***Abstract Code***

* User clicked on ***Cancel*** button from either **Search Resource Results**
* Get Incident.ID and Resouce.ID and User.ID from session query.
* Run the **Delete Request Task** task; remove resource request
  + From Request table delete request where Request.ID is a composed of Incident.ID and Resource.ID. and Request.Requested\_by=CurrentUser,

1. ***Reject Request <-needs a lot of work. Do we want to delete the request when rejected or update status to rejected?***

***Task Decomposition***

***Lock Types:*** Update to Requests Table

***Number of Locks:*** 2 db locks

***Enabling Conditions:*** Enabled by a user login and clicking **Reject** button

***Frequency:*** Several times a day

***Consistency***: DB needs to be updated correctly. Critical

***Subtasks:*** None

***Decomposition****: Not* Required

***Abstract Code***

* User clicked on ***Reject*** button from either **Search Resource Results**
* Get Incident.ID and Resouce.ID and User.ID from session query.
* Run the **Delete Request Task** task; remove resource request
  + From Request table delete request where Request.ID is a composed of Incident.ID and Resource.ID. and Request.Requested\_by=CurrentUser,

1. ***Return Resource <-needs a lot of work abc***

***Task Decomposition***

***Lock Types:*** Update to Requests table, Update to Resources Table

***Number of Locks:*** 2 db locks

***Enabling Conditions:*** Enabled by a user login and clicking **Return** button

***Frequency:*** Several times a day

***Consistency***: DB needs to be updated correctly. Critical

***Subtasks:*** None

***Decomposition****: Not* Required

***Abstract Code***

* User clicked on ***Return*** button from either **Search Resources**
* Get Incident.ID and Resouce.ID and User.ID from session query.
* Run the **Update Resource Availability Task** task; update resource information
  + From Resources select resources that that was selected.
  + If Resource.Owner == CurrentUser
    - Update Resource.Status to “Available”
  + Else reject DB update
* Run the **Update Request Task** task; update request information
  + From Request table select Request where Request.Resource AND Request.Incident== CurrentResource AND Current Incident
  + If Request.Resource.Owner=CurrentUser:
    - Update Request.Status to “RETURNED”
  + Else
    - Reject DB Update.

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

1. *View Resource Report*
   1. *Lookup*
   2. *Medium portion of DB*
   3. *?*
   4. *?*
   5. *NO*
   6. *NO*
   7. *LEAVE AS IS*
2. *Resource Status*
   1. *Lookup, Insert, Update*
   2. *Large portion of DB*
   3. *?*
   4. *NO*
   5. *YES*
   6. *YES*
   7. *DECOMPOSE -> Lookup incidents, return resource, approve resource, deploy resource, reject resource request*

**Abstract Code:**