**Team: Smiling Solo**

**Project: Patch Request Application**

**Project Specifications**

**Requirements**

A Patch Request Application keeps track of all patch requests of an engineering group, including the bug number, target date, and priority. The patch request will be posted for a specific software branches, and single patch may have multiple bug fixes.

Only engineers may login to the system, however, even without logging in, a user can get the list and contents of the released patches.

The engineers can adjust the priority and target date, and can see the pending patches. Developers register themselves to fix the patch requests, and provide the estimate fix schedule.

Every patch request except locally found bugs has customer list of who are waiting for the bug fix, and the dollar number for the impact. For the internally found bugs, the customer is set to ‘local’ and the potential cost will be assigned instead of dollar number for the impact. When a project is lost, the relevant bugs of the lost customer will be re-evaluated to re-priorities the bug fix schedule.

Patch request may be assigned to an unscheduled pending patch, or specified patch. Every branch should have at least one unscheduled patch of which name is not defined.

Develop manager schedule a patch and give the name and target date based on the priorities of the included patch requests. Developer will announce the patch with release date.

**Entity-Relationship Diagram**

**Entity Classes & Constraints**

PatchRequest

* Every patch request is assigned to a patch.

Engineer

* Every engineer has a name

TacSupport

* Every engineer has a name

Developer

* Every engineer has a name

Customer

* Every customer has name and revenue value

Branch

* Every branch has its name and frs, eor, eol dates

Patch

* Every patch has the branched

**Relational Schema Diagram**

**User Role Diagram**

**User Operations**

\* indicates operations to be implemented

**User**

Query

\*ShowAllBranches: Show the names & FRS, EOE, EOL date of all branched

\*ShowAllPatches( :bid ): Show the names & release date, included bug is of patches of a branch. If public, shows only released patches

**Unknown User**

Actions

\*Login( :engid, :pwd ): Login the specified user

**Engineer**

Queries

\*ShowRequests( :pid ): Show the patch requests of an patch

\*GetMyInfo: Get my personal information

Actions

\*PostRequests( :bugid, :priority, :tdate, :bid ): Post a patch request

\*SetPwd( :oldpwd, :newpwd, :confpwd ): Changes password

\*Logout: Logout the current user

**Support Engineer**

Actions

\*ChangeRequests( :priority, :tdate ): Set/change priority and the target date of the patch request

**Developer**

Actions

\*SetEstimateDate( :prid, :edate ): Set estimate fix date

\*AnnouncePatch( :pid, :rdate ): Publish a patch

SetDeveloperCategory( :engid, :category ): Set developers development area

SetEngineerRole( :engid, :erole ): Set the role of an engineer

**DevManager**

Queries

\*ShowDeveloperCategory: Show developers with category

Actions

\*SetDeveloper( :prid, :dengid ): Assign a developer to fix the patch request.

\*SchedulePatch( :pid, :pname, :tdate ): Create a patch item.

\*CreateBranch( :bname ): Crate a new software branch.

\*AddRequest( :pid, :prid ): Add a patch request to a patch

\*DelRequest( :pid, :prid ): Delete a patch request from a patch

**Actions with State-Related Post-Conditions**

AddRequest

* The target date of the patch will be adjusted by the sum priority of the request items

**Conceptual Transition Constraints**

🡺 SetActMgr

Constraint: If a department loses its dept manager or acting manager, or if an empty department gains new employees (but not a department manager), the employee in the department with the highest salary (at the end of the base operation), or one such employee if there are more than one, becomes the acting manager.

Operations: AddEmp, ChangeJob, ChangePosition, TerminateEmp

🡺 OnLoseDept

Constraint: When a department is deleted, all employees in the department are either terminated, or moved together to a different department

Operations: DestroyDept

🡺 OnLoseMgr

Constraint: When a manager is terminated, all employees who directly reported to them end up directly reporting to their manager

Operations: TerminateEmp, DestroyDept

**Operations with Identity-Based User Pre-Conditions**

Login

* a user can’t login if already logged in

AddEmp

* a dept mgr can only hire an employee into their dept, and if their initial direct manager is in the same dept

ChangeJob

* an employee can't change their own job
* a dept mgr can only change the jobs of employees in their own department

ChangeSal

* an employee can't change their own sal
* a dept mgr can only change the salaries of employees in their own department

ChangePosition

* an employee can't change their own job, mgr or dept
* a dept mgr can only change the jobs of employees in their own department
* a dept mgr can only change the mgr of employees in their own department, and then only if both the old and new mgr are in their dept as well
* a dept mgr cannot change an employee's dept

TerminateEmp

* an employee can't terminate themselves
* a dept mgr can only terminate employees in their own dept

**Queries with Output-Related Post-Conditions**

ShowAddr

* Only shows employees who allow their address to be public

ShowCtrlEmps

* If the user is a dept mgr, only shows information about employees in their department.

**Conceptual Access Constraints**

**Informational Status Access Constraints**

🡺 SeeAddr   
An employee can see another employee's address only if they have made it public

**Causal Status Access Constraints**

None

**Informational Role-Based Access Constraints**

🡺 EmployeeQueries

All queries except ones that just provide the numbers, names & locations of departments require that the user be logged in (i.e. their role or some ancestor role must be Employee)

**Causal Role-Based Access Constraints**

🡺 EmployeeActions

All actions except those that login a user require that the user be logged in (i.e. their role or some ancestor role must be Employee)

🡺 AffectDept

An employee's department can only be changed by the President

🡺 PrezActions

A department can only be created, destroyed or moved by the President

**Informational Identity-Based Access Constraints**

🡺 SeeSal   
An employee's salary can only be seen by the President, or by their department manager

🡺 SeeJobMgr   
Users who are not executives can only find out the job or mgr of employees in their department, or that they directly or indirectly manage

**Causal Identity-Based Access Constraints**

🡺 GoodLogin   
A user cannot login if that user is already logged in

🡺 ChangeOwnAddr

An employee's address can only be changed by themselves

🡺 ChangeOwnPwd

An employee's password can only be changed by themselves

🡺 AffectSelf

An employee cannot change their own salary, job, manager or dept

🡺 AffectPrez

No operation may be used to hire, terminate or change the salary, direct manager, dept or job (new or old) of the president

🡺 NewMgr

An employee can be hired by a dept manager only if their new direct manager is in the department manager's department.

🡺 ExecutiveActions

An employee can only be hired, fired or have their job, salary or manager changed by the President, or by the manager of their department

🡺 UpdateMgr

An employee's direct manager can only be changed by a department manager if both the old and new direct manager are in the department manager's department.

**Complex Conceptual State Constraints Enforced by Check Constraints**

🡺 NoPrezMgr  
The President does not have a manager

In Emps, CHECK( job <> 'PRESIDENT' OR mgr IS NULL )

🡺 EmpMgr  
Every employee other than the President has a manager

In Emps, CHECK( job = 'PRESIDENT' OR mgr IS NOT NULL )

**Complex Conceptual State Constraints Requiring Other Enforcement**

🡺 OnePrez

Constraint: Only one employee has the job of president

Operations: AddEmp, ChangeJob, ChangePosition

Tables: Emps

🡺 ManageCirc

Constraint: No one manages themselves, their manager, their manager's manager, etc.

Operations: AddEmp, ChangeMgr, ChangePosition

Tables: Emps

🡺 ManageDeptMgr

Constraint: All department managers are directly managed by the President (if there is one, or by no one if there isn't)

Operations: AddEmp, ChangeMgr, ChangeJob, ChangePosition

Tables: Emps

🡺 OneDeptMgr

Constraint: Every department with employees either has one department manager or an acting manager who works for that dept, but not both

Operations: AddEmp, ChangeJob, ChangePosition, TerminateEmp

Tables: Emps

🡺 DeptMgrClerks

Constraint: Every department that has a clerk has a dept manager

Operations: AddEmp, ChangeJob, ChangePosition, TerminateEmp

Tables: Emps

🡺 ManageLoc

Constraint: Every employees works at the same location as their direct manager, if they have one, unless their direct manager is the president

Operations: AddEmp, ChangeMgr, ChangePosition

Tables: Emps, Depts

**Manifest Views (and Relational State Assertions)**

🡺 OnePrez

Only one employee has the job of president

Manifest View: NONE

State Assertion:  
(SELECT count(\*) FROM Emps WHERE job = 'PRESIDENT') <= 1

🡺 ManageCirc

No one manages themselves, their manager, their manager's manager, etc.

Manifest View:

-- provide a view with pairs of each employee and the manager at every level

CREATE RECURSIVE VIEW Mgrs( empno, mgr ) AS

SELECT empno, mgr FROM Emps WHERE mgr IS NOT NULL

UNION

SELECT e.empno, m.mgr FROM Emps e, Mgrs m WHERE e.mgr = m.empno

State Assertion:

NO Mgrs SATISFIES empno = mgr

🡺 ManageDeptMgr

All department managers are directly managed by the President,

or by no one if there is no President

Manifest View: None

State Assertion:

WITH VALUE thePrez AS (SELECT empno FROM Emps WHERE job = 'PRESIDENT')  
CASE (thePrez IS NULL) THEN

EACH Emps WHERE job = 'DEPTMGR' SATISFIES mgr IS NULL

ELSE

EACH Emps WHERE job = 'DEPTMGR' SATISFIES mgr = thePrez

END CASE

🡺 OneDeptMgr

Every department with employees either has one department manager or an acting manager who works for that dept, but not both

Manifest View:

-- For each department, provides count of #employees, actmgrs & deptmgrs

CREATE VIEW DeptMgrsView AS   
SELECT deptno,

(SELECT count(\*) FROM Emps e  
 WHERE e.deptno = d.deptno) AS eknt,  
 (SELECT count(\*) FROM Emps e  
 WHERE e.deptno = d.deptno AND e.empno = d.actmgr) AS aknt

(SELECT count(\*) FROM Emps e  
 WHERE e.deptno = d.deptno AND e.job = 'DEPTMGR') AS mknt  
FROM Depts d

State Assertion:

EACH DeptMgrs d WHERE eknt > 0 SATISFIES aknt + mknt = 1

State Assertion without Manifest View:

EACH Depts d WHERE (SOME Emps e SATIFIES e.deptno = d.deptno)

SATISFIES

(SELECT count(\*) FROM Emps e  
 WHERE e.deptno = d.deptno AND e.empno = d.actmgr) +

(SELECT count(\*) FROM Depts d  
 WHERE e.deptno = d.deptno AND job = 'DEPTMGR') = 1

🡺 DeptMgrClerks

Every department that has a clerk has a dept manager

Manifest View:

-- For each dept, has # of clerks (cknt) & # of deptmgrs (mknt)

CREATE VIEW DeptClerkMgrs AS   
SELECT deptno,

(SELECT count(\*) FROM Emps e  
 WHERE e.deptno = d.deptno AND e.job = 'CLERK') AS cknt

(SELECT count(\*) FROM Emps e  
 WHERE e.deptno = d.deptno AND e.job = 'DEPTMGR') AS mknt

FROM Depts d

State Assertion:

EACH DeptClerkMgrs d WHERE d.cknt > 0 SATISFIES d.mknt > 0

State Assertion without Manifest View:

EACH Depts d  
WHERE (SOME e WHERE e.deptno = d.deptno SATSIFIES e.job = 'CLERK')  
SATISFIES (SOME e WHERE e.deptno = d.deptno SATISFIES e.job = 'DEPTMGR')

🡺 ManageLoc

Every employees works at the same location as their direct manager, if they have one, unless their direct manager is the president

Manifest View:

-- for each employee with a mgr, indicates their job, dept and their dept location

-- their mgr, and their mgr's job, dept, and dept location

CREATE VIEW MgrLocsView AS  
 SELECT e.empno AS eno, e.loc AS eloc,  
 m.empno AS mno, m.loc AS mloc, m.job AS mjob  
 FROM EmpsDept e, EmpsDept m WHERE e.mgr = m.empno

State Assertion:

EACH MgrLocsView WHERE mjob <> 'PRESIDENT'  
 SATISFIES mloc = eloc

State Assertion without Manifest View:

WITH EmpsDept AS (

SELECT empno, mgr, loc, job FROM Emps NATURAL JOIN Depts)

VALUE thePrez AS (SELECT empno FROM Emps WHERE job = 'PRESIDENT')  
EACH EmpDepts e WHERE e.mgr <> thePrez AND e.mgr IS NOT NULL

SATISFIES e.loc = (SELECT loc FROM EmpDepts m WHERE e.mgr = m.empno)