Mercury - A Laboratory Information and Management System

Todd Merport
Alexander Proskurowski
Katalin Voros

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-149

http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-149.html

August 13, 2014

UNIVERSITY OF CALIFORNIA, BERKELEY

CITRIS -
THE CENTER FOR
INFORMATION TECHNOLOGY
RESEARCH IN THE INTEREST OF SOCIETY

MARVELL NANOFABRICATION LABORATORY
520 SUTARDJA DAI HALL
BERKELEY, CALIFORNIA 94720-1754
TEL: 510-809-8600

Mercury

A Laboratory Information and Management System

Todd Merport
Alexander Proskurowski
Katalin Voros

Abstract

Mercury is a computer management and information system designed specifically to operate the Berkeley Marvell NanoLab and to use laboratory resources efficiently. The components of Mercury are a relational database management system (Ingres), daemons (background processes), and clients, in a dual, three tier application. This report describes the system design, database details, implementation technologies, and it gives some detailed examples. The Mercury system has been in real-time operation since December 2009 in the UC Berkeley NanoLab.

2014

Table of Contents

List of Figures

Glossary

Apache Tomcat	Open source software implementation of the Java Servlet and JavaServer Pages
BCIMS	Berkeley Computer-Integrated Manufacturing System
BIRT	Business Intelligence Reporting Tool
CAPE	Common and Personal Environment
CMOS	Complementary Metal-Oxide-Silicon (transistor or process)
daemon	A computer program that runs as a background process, rather than being under the direct control of an interactive user.
EECS	Department of Electrical Engineering and Computer Sciences
GPIB	General Purpose Interface Bus (Hewlett-Packard)
GUI	Graphical User Interface
HYDRA	Equipment control system in the NanoLab
IceFaces JSF	Java development platform (JavaServer Faces 2)
Java	A multifaceted programming language
LN	liquid nitrogen
LPCVD	Low Pressure Chemical Vapor Deposition
MERCURY	Name of the NanoLab's management and information system
MERCURY CLIENT	Program to execute tasks in the NanoLab
MERCURY WEB	Web interface to Mercury system
RDBMS	Relational Database Management System
Remote desktop	A PC connected to the remote server CAPE
RUMS	Resource Utilization Monitoring System (environmental sensors)
Poly-Si	Polycrystalline silicon
SQL	Structured Query Language
SQLServer	relational database management system developed by Microsoft
UNIX	operating system

I. Introduction and Overview

Mercury is a computer management and information system designed specifically to operate the Berkeley Marvell NanoLab and to use laboratory resources efficiently. Mercury was developed parallel to the building of the new NanoLab, same as was in the case of its predecessor, the Microlab 30 years before [1]. However, facilities development and computer systems design of the new lab (2010) had the advantage of the experience of the old systems behind them. The Berkeley Computer Integrated Manufacturing System (BCIMS) [2] had 25 years of real-time experience in constant use, with over two million activities captured and many enhancements added, upgrading, and fine tuning along the way, provided a solid foundation for the design of Mercury. The software is named Mercury (not an acronym) after the Roman god that acted as a messenger.

The components of Mercury are a relational database management system (Ingres), daemons (background processes), and clients. Mercury is a dual, three tier application (see Fig. 1). The client program that runs in the laboratory is called Mercury Client. It connects to a session management daemon, Mercury Server. There is also another system, called Mercury Web, which provides a web interface to the system and runs under Apache Tomcat. Most of the logic or business rules for the system are implemented in the database as stored procedures. This helps insure data integrity and improves speed. It also minimizes duplication of procedures in the middle tier and clients.

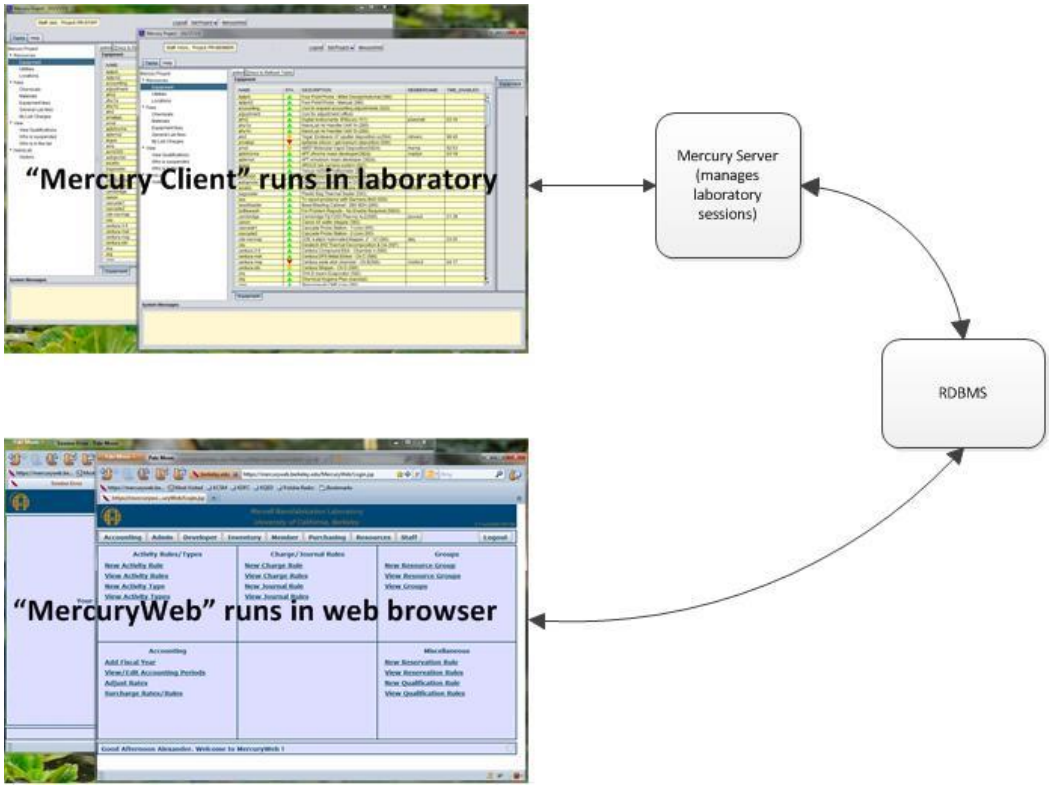

"Mercury Client" runs in laboratory
Mercury Server (manages laboratory sessions)
RDBMS
"MercuryWeb" runs in web browser

Fig. 1. Mercury structure, a dual three tier application system.

Mercury emphasizes accounting by utilizing a double entry accounting system. Activities are recorded and debited/credited to the appropriate accounts. The system maintains a real-time state of accounts in contrast to monthly and annual reporting. Further, all transactions in the laboratory are managed using database transactional facilities. These transactions include lab entries, equipment usage, and inventory. When members use or manage resources, activities are generated and information is stored in the journal and ledger. (Fig. 2.)

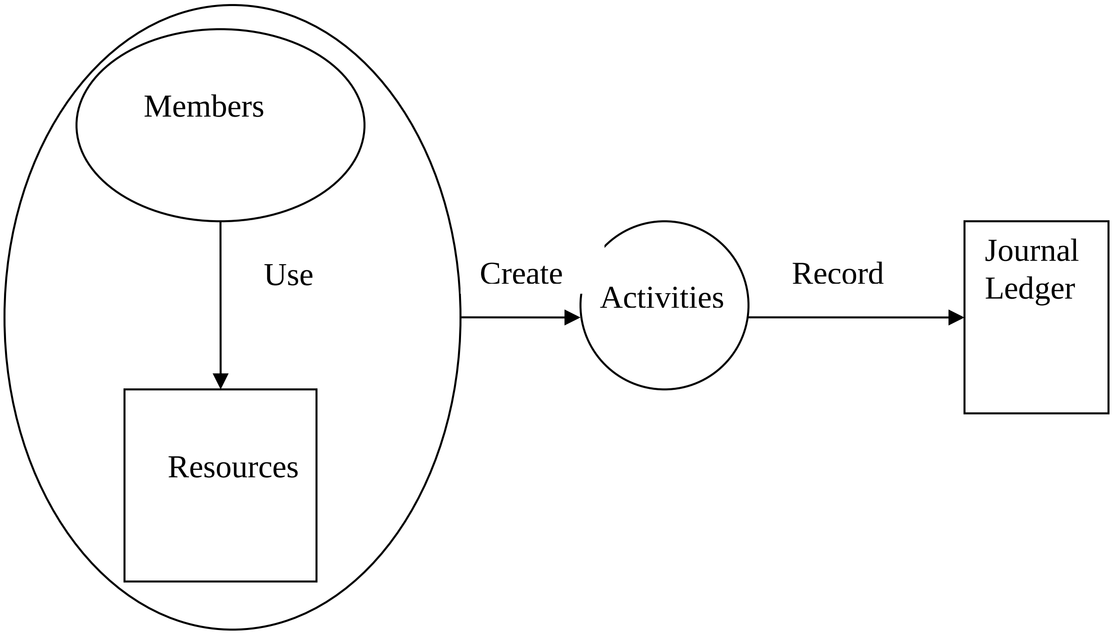


Fig. 2. Creation of an accounting activity.

This report describes the system as it is in operation in the Berkeley Marvell NanoLab, in 2014, including database design, equipment communication and control, activities and recharge accounting, and at first, users' (laboratory members'), computer (Mercury system) interaction experience in the lab.

II. User Experience

The Marvell NanoLab at Berkeley is a graduate student semiconductor research facility with full IC processing capability, involving over 130 pieces of equipment. Students and associated researchers (collectively called lab members), numbering more than 300 in any given month, work on their own individual projects on a selected number of tools in diverse combinations. Protection from cross-contamination is a major concern, along with tool accessibility and scheduling. Mercury enables the control of this complex environment and allows charging various research grants according to general laboratory and special equipment use. Computers are part of the facility and running a semiconductor device process requires computer interaction every step of the way.

New lab members go through an introductory seminar, which includes information on the computer system and instructions on its use. First an overview is given of the layers of interaction, shown in Fig. 3. The user (lab members, staff, and management) has access to the lab environment through its computer control system, Mercury. Another way to look at this is that the Mercury computer system, built modularly, connects the equipment, environmental controls, and user activities with a common, relational database. Information from the database is then utilized to carry out activities, such as enabling equipment for processing or providing reports.

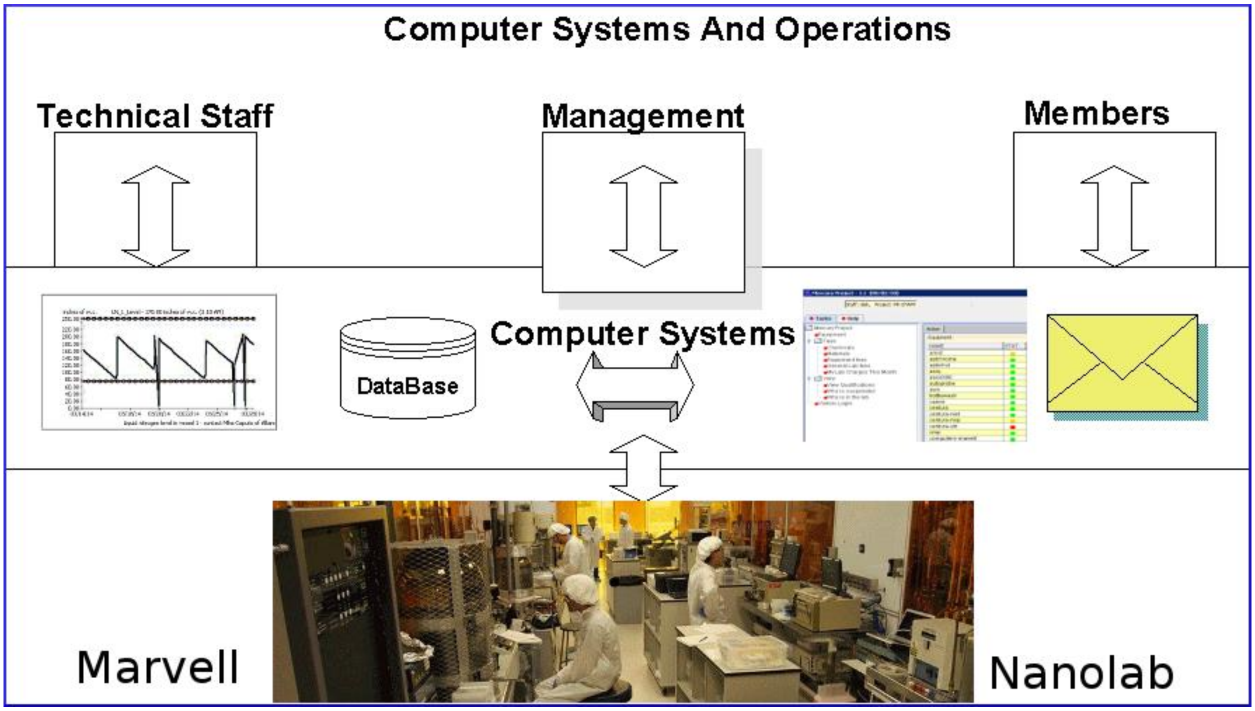


Fig. 3. Schematic outline of computer interactions in the Marvell NanoLab.

CAPE

Upon entering the lab the first step is to login to CAPE (Common And Personal Environment) from a PC in the lobby (Fig. 4), by clicking on the CAPE icon on the screen (Fig. 5.) CAPE is a desktop system (Windows Terminal Server) allowing access to Mercury Client, the equipment controller program in the lab and to restricted internet access through a web browser.

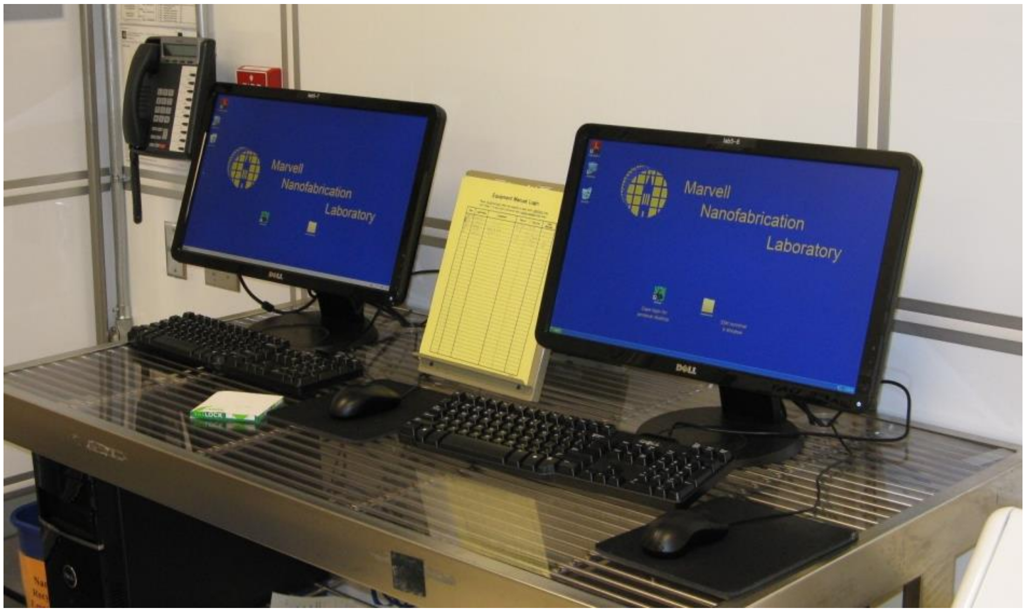


Fig. 4. Login terminals in the entrance lobby of the NanoLab.

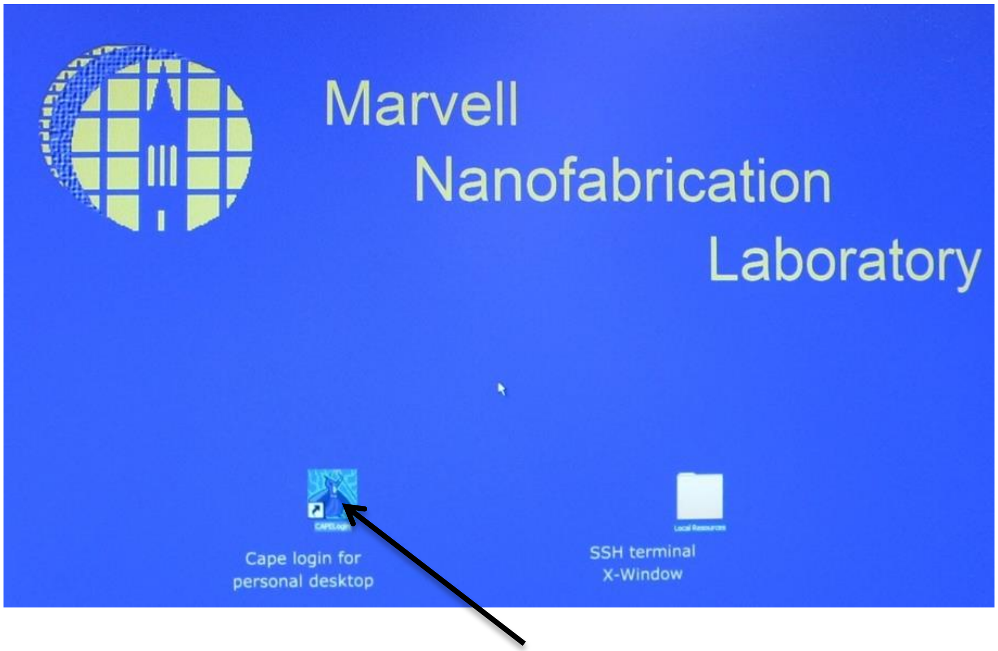


Fig. 5. Click on the CAPE icon to login.

Once logged in, the user will click on the Mercury (Client) icon to start a lab session (Fig. 6.) This lets the system and other people inside know that she/he is in the lab. At that point the user is allowed to select one of several projects, if this is the case, to which charges are to be made, and accounting functions are initialized.

The Mercury Client (Fig. 8) shows the state of the equipment by status indicators. If green, the lab user is permitted to enable and disable equipment. The user may also start the browser for internet access. This is useful for information retrieval, such as lab manuals, while working inside the lab.

To move to another area in the NanoLab, the user clicks on the HIDECAPE icon. The CAPE session is now suspended, which means that he is still logged into the lab, and the session can be reopened at another terminal inside. At another terminal in the lab, after login to CAPE again the session is reactivated, and the user will see his desktop as he left it.

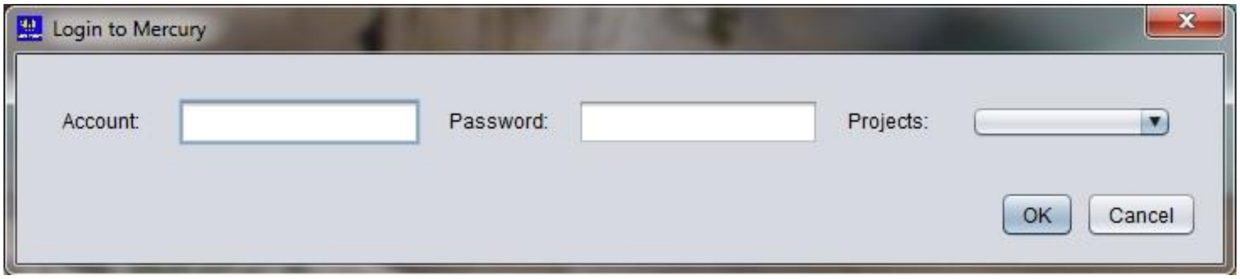
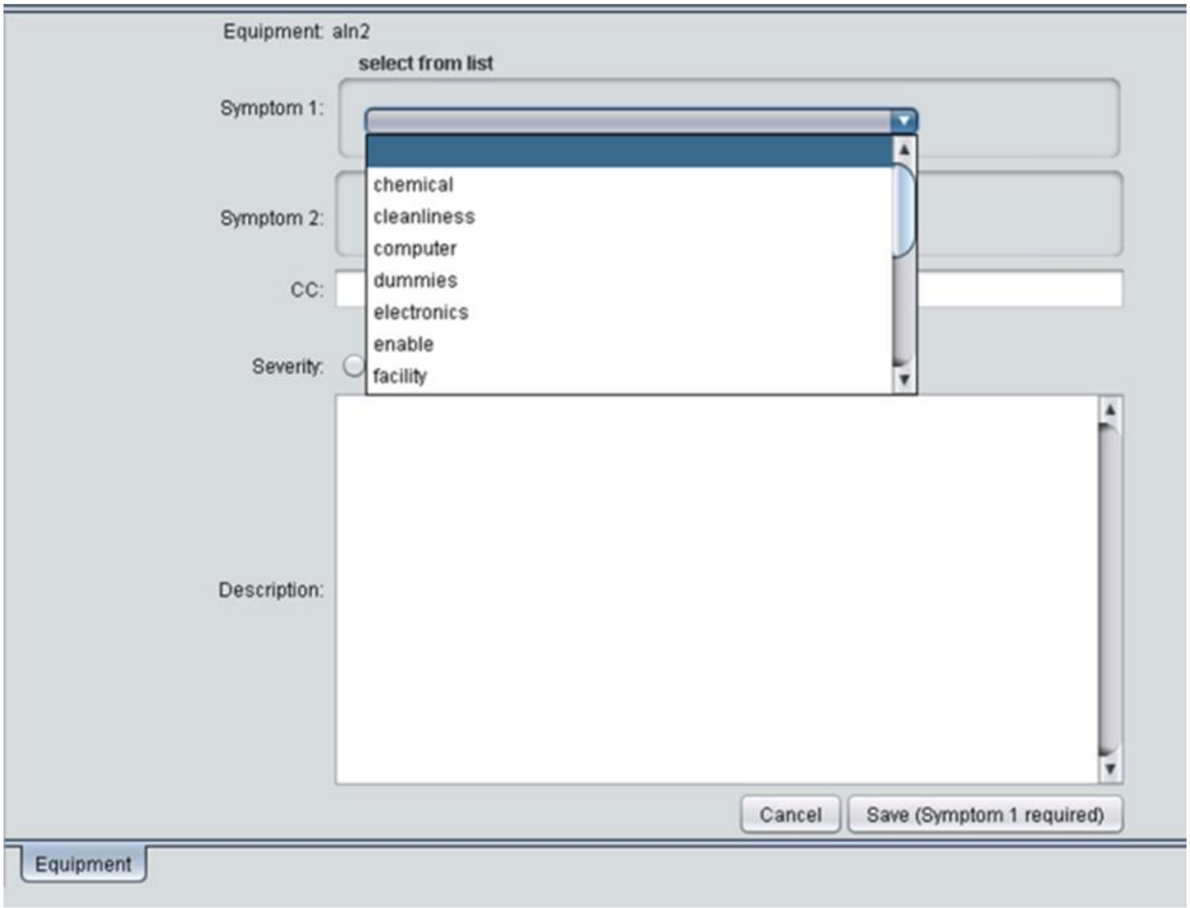


Fig. 6. Mercury (Client) login window with option to select a project from a drop-down menu.

Mercury Client is available only from CAPE sessions started in the NanoLab, to use it in conjunction with equipment in the lab. This means that equipment cannot be enabled /disabled remotely from non-designated locations, and that the user has to be present for any equipment activity. This also means that the user is charged laboratory use fees as long as he is logged into Mercury Client. Satellite labs in other locations (Cory Hall, Surtardja Dai Hall) also can invoke a CAPE session, but members can only enable equipment associated with the terminal (or client) location. For certain process steps, such as long furnace runs, the user may leave and not be charged lab fees; however, equipment fees are accrued as long as the equipment is enabled.

Mercury Client

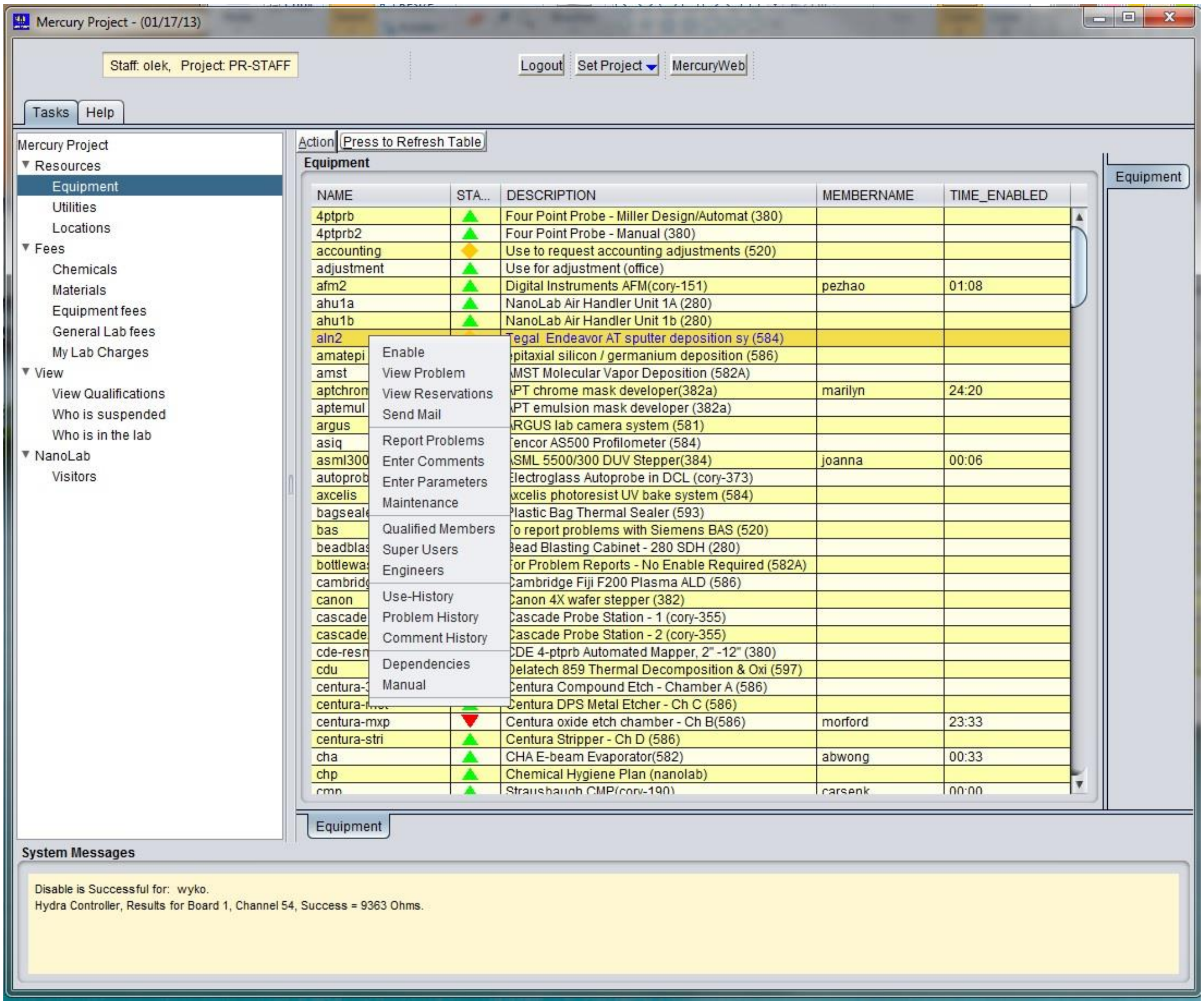
Clicking on a tool name a pop-up menu presents several action possibilities (inset in Fig. 8). These are: enable/disable; view/report problems; send mail to all the qualified users of that tool; view reservations; enter comments and process parameters; viewing of several important aspects of tool operation, such as tool repairs and maintenance schedules, what utilities are needed to operate it (dependencies), who used it previously, comments/problem history; members who are qualified to use it, and the operation manual of the tool. All tool-specific actions are entered here and maintained in the database.



Equipment: aln2

Fig. 7. Problem reporting page of Mercury Client.

Clicking on the Report Problem option the user is presented with the page shown in Fig. 7. This module is based on FAULTS: An Equipment Maintenance and Repair Tracking System Using a Relational Database, described in detail in Reference [4]. The module requires that the user select the appropriate description of the problem from a list of symptoms. He can detail the problem by adding text in the Description window. Upon hitting the Save button the status indicator on the front page (Fig. 8) turns yellow or red, depending on the severity of the problem and an email is sent automatically to the equipment engineer responsible for the tool. Upon completing the repair the engineer clears the problem by selecting the appropriate fault from the menu shown, and enters, in the comments section, details of the repair. At that point the problem report is cleared and the status indicator switches to green.



Equipment

NAME	STA...	DESCRIPTION	MEMBERNAME	TIME_ENABLED
4ptprb	▲	Four Point Probe - Miller Design/Automat (380)		
4ptprb2	▲	Four Point Probe - Manual (380)		
accounting	◆	Use to request accounting adjustments (520)		
adjustment	▲	Use for adjustment (office)		
afm2	▲	Digital Instruments AFM(cory-151)	pezhao	01:08
ahu1a	▲	NanoLab Air Handler Unit 1A (280)		
ahu1b	▲	NanoLab Air Handler Unit 1b (280)		
aln2		Tegal Endeavor AT sputter deposition sy (584)		
amatepi		pitaxial silicon / germanium deposition (586)		
amst		MST Molecular Vapor Deposition (582A)		
aptchrom		PT chrome mask developer(382a)	marilyn	24:20
aptemul		PT emulsion mask developer (382a)		
argus		RGUS lab camera system (581)		
asiq		encor AS500 Profilometer (584)		
asml300		SML 5500/300 DUV Stepper(384)	joanna	00:06
autoprob		lectroglass Autoprobe in DCL (cory-373)		
axcelis		xcelis photoresist UV bake system (584)		
bagseale		Plastic Bag Thermal Sealer (593)		
bas		o report problems with Siemens BAS (520)		
beadblas		ead Blasting Cabinet - 280 SDH (280)		
bottlewa		or Problem Reports - No Enable Required (582A)		
cambrid		ambridge Fiji F200 Plasma ALD (586)		
canon		anon 4X wafer stepper (382)		
cascade		ascade Probe Station - 1 (cory-355)		
cascade		ascade Probe Station - 2 (cory-355)		
cde-resn		CDE 4-ptprb Automated Mapper, 2" -12" (380)		
cdu		elatech 859 Thermal Decomposition & Oxi (597)		
centura-3		entura Compound Etch - Chamber A (586)		
centura-met	▲	entura DPS Metal Etcher - Ch C (586)		
centura-mxp	▼	Centura oxide etch chamber - Ch B(586)	morford	23:33
centura-stri	▲	Centura Stripper - Ch D (586)		
cha	▲	CHA E-beam Evaporator(582)	abwong	00:33
chp	▲	Chemical Hygiene Plan (nanolab)		
cmp	▲	Straushaugh CMP(cory-190)	carsenk	00:00

Disable is Successful for: wyko.
Hydra Controller, Results for Board 1, Channel 54, Success = 9363 Ohms.

Fig. 8. Mercury Client session viewing equipment.

The bottom part of the Mercury Client interface shows equipment charges incurred during the session and other system messages.

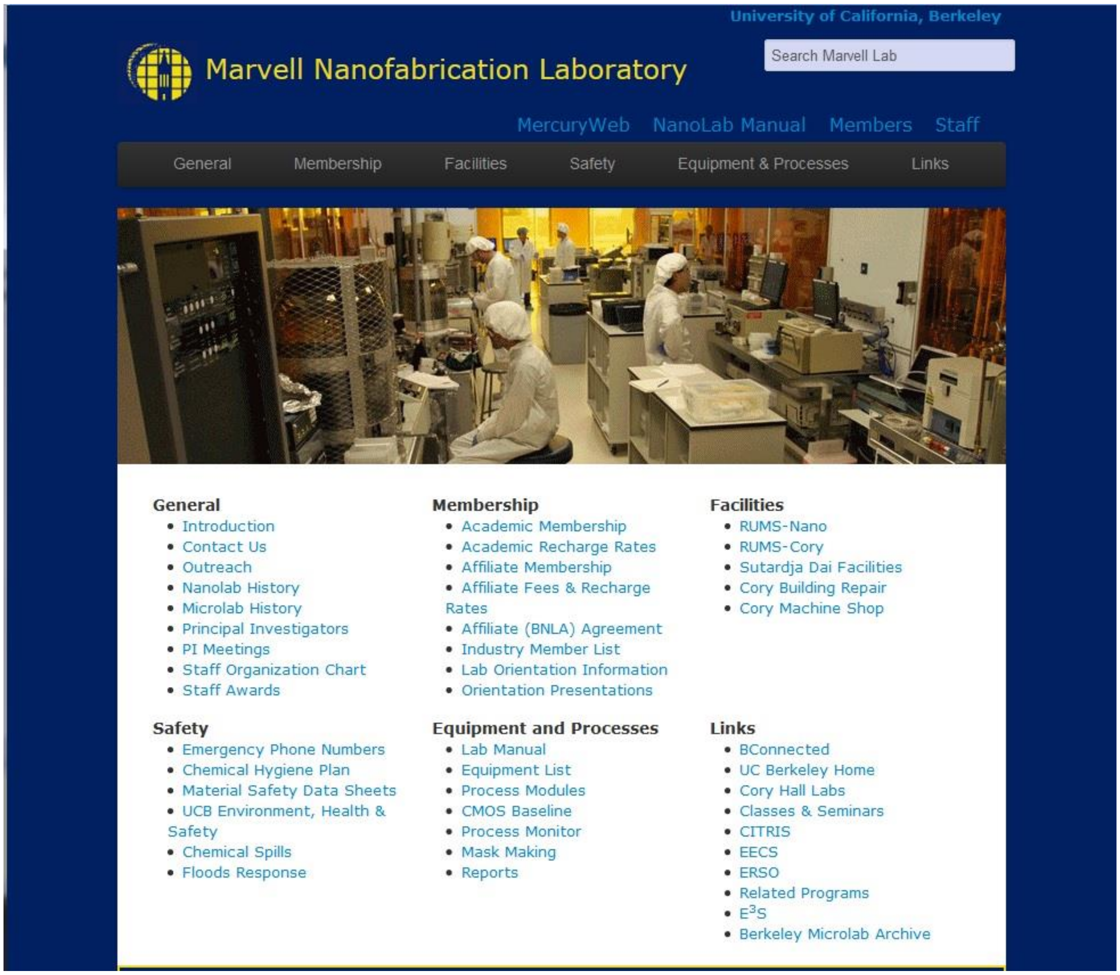
The left side (or tree) of the Mercury Client interface allows members to select a general category that is detailed in the right side table list (table). The right side lists can stack similarly to a way a spreadsheet or browser has tabs to indicate hidden sheets.

The left side of the Mercury Client interface provides useful information to lab members inside the lab. This includes equipment restrictions listings, utilities and locations, information on lab charges, materials and chemical availability, equipment qualifications listings, who is in the lab and who is suspended. There is also an option to login visitors.

The Mercury Client session will stay active until the user clicks on the Logout button in the Mercury Client window (when leaving the NanoLab). This will log him out of the NanoLab.

Mercury Web

Mercury Client has a companion web site called Mercury Web, accessible both through the Marvell NanoLab web site (Fig. 9) and the Mercury Client (Fig. 7.) Mercury Web allows lab members to make reservations, view inventory, check equipment status, see who is in the lab, and do various other tasks that may be done outside the laboratory and will not be charged lab fees. Mercury Web can be used from any remote location including from home.



General
- Introduction
- Contact Us
- Outreach
- Nanolab History
- Microlab History
- Principal Investigators
- PI Meetings
- Staff Organization Chart
- Staff Awards

Membership
- Academic Membership
- Academic Recharge Rates
- Affiliate Membership
- Affiliate Fees & Recharge Rates
- Affiliate (BNLA) Agreement
- Industry Member List
- Lab Orientation Information
- Orientation Presentations

Facilities
- RUMS-Nano
- RUMS-Cory
- Sutardja Dai Facilities
- Cory Building Repair
- Cory Machine Shop

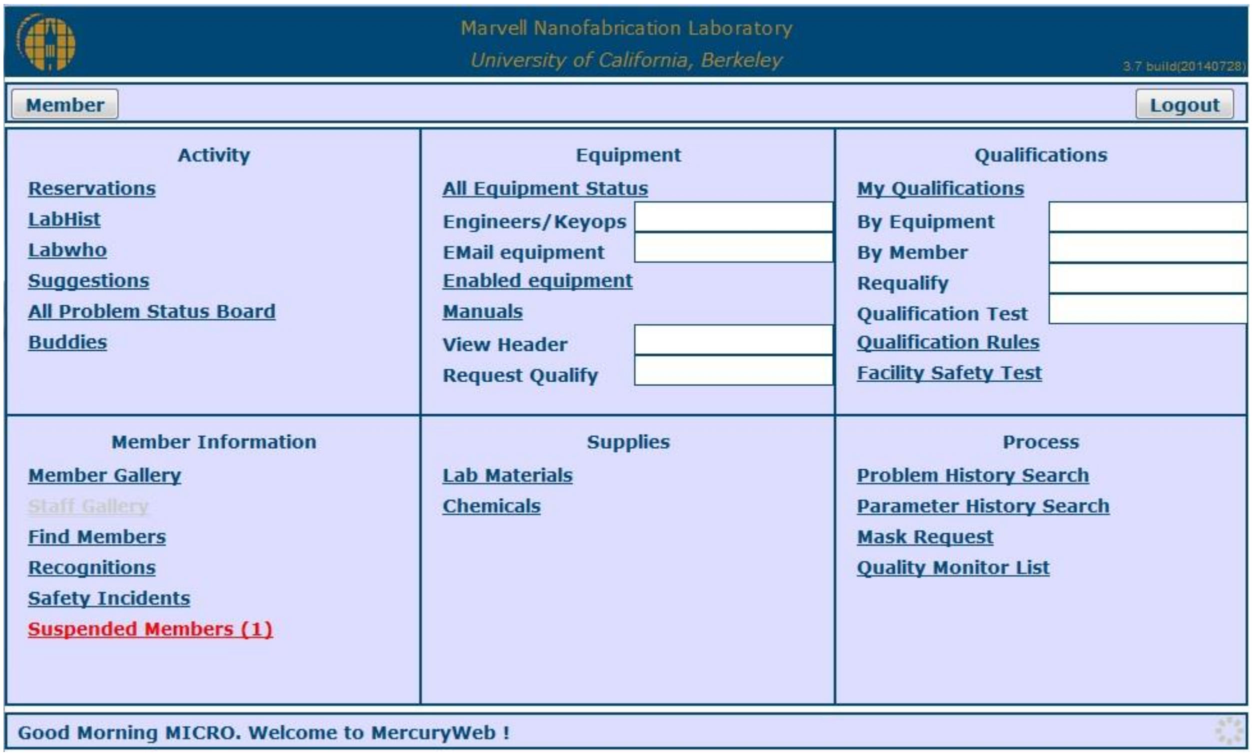
Safety
- Emergency Phone Numbers
- Chemical Hygiene Plan
- Material Safety Data Sheets
- UCB Environment, Health & Safety
- Chemical Spills
- Floods Response

Equipment and Processes
- Lab Manual
- Equipment List
- Process Modules
- CMOS Baseline
- Process Monitor
- Mask Making
- Reports

Links
- BConnected
- UC Berkeley Home
- Cory Hall Labs
- Classes & Seminars
- CITRIS
- EECS
- ERSO
- Related Programs
- E^{3}S
- Berkeley Microlab Archive

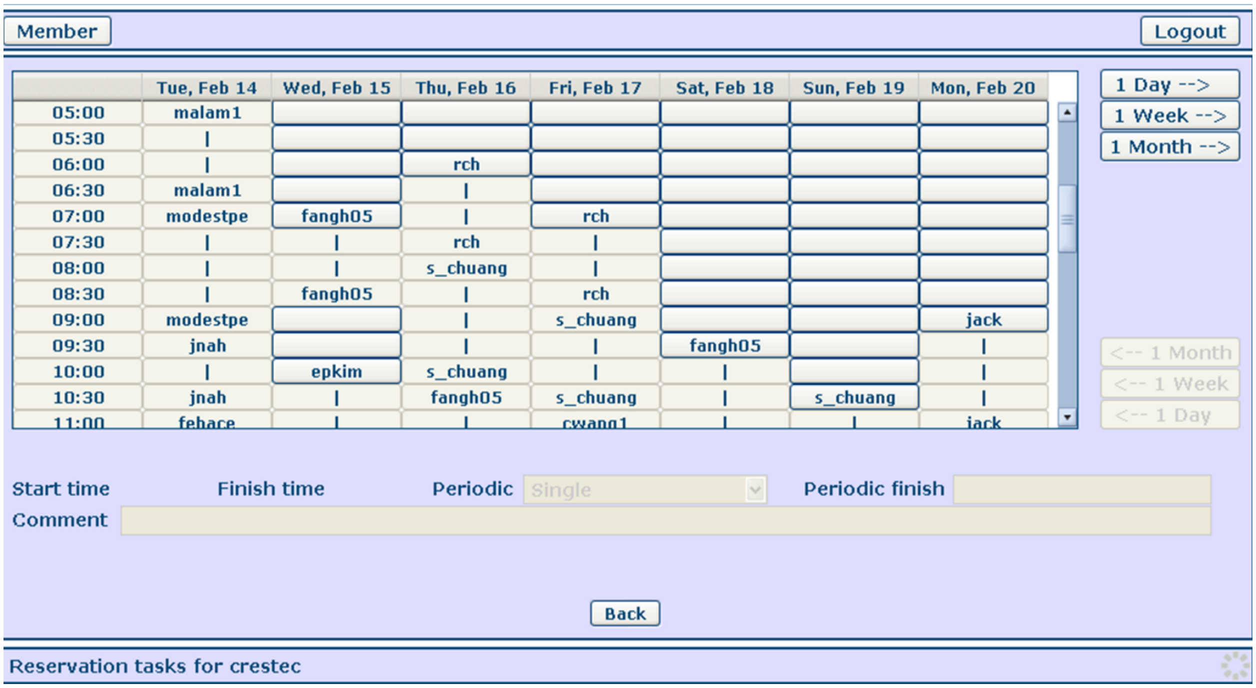
Fig. 9. Marvell NanoLab web portal (partial view), with access to Mercury Web.

When clicking on the Mercury Web button (on top of the web portal, Fig. 9) and logging in, the appropriate Mercury Web page will appear, depending on classification: lab member, staff, management. Fig. 10 shows the member page, with available actions for lab members. This is where equipment reservations can be made, check what equipment is being used currently, see the equipment status board, information obtained about lab members and who is qualified on what machine. The Member Information is useful when looking for an equipment training partner. Fig. 11 shows the equipment reservations page.



Good Morning MICRO. Welcome to MercuryWeb !

Fig.10. Mercury Web member page



	Tue, Feb 14	Wed, Feb 15	Thu, Feb 16	Fri, Feb 17	Sat, Feb 18	Sun, Feb 19	Mon, Feb 20
05:00	malam1						
05:30	|						
06:00	|		rch				
06:30	malam1		|				
07:00	modestpe	fangh05	|	rch			
07:30	|	|	rch	|			
08:00	|	|	s_chuang	|			
08:30	|	fangh05	|	rch			
09:00	modestpe		|	s_chuang			jack
09:30	jnah		|	|	fangh05		|
10:00	|	epkim	s_chuang	|	|		|
10:30	jnah	|	fangh05	s_chuang	|	s_chuang	|
11:00	fehace	|	|	cwang1	|	|	jack

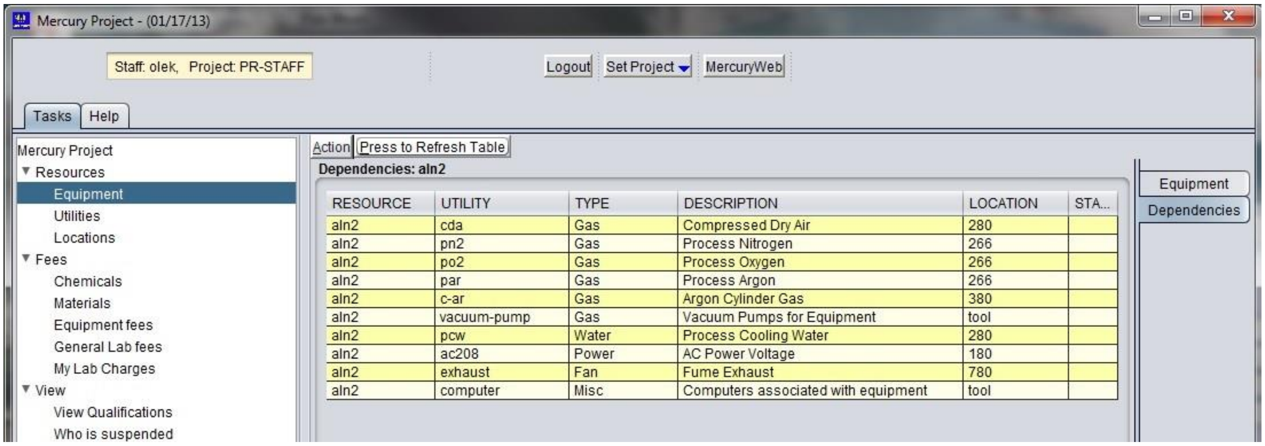
Reservation tasks for crestec

Fig. 11. Mercury Web equipment reservation page.

Additional Informational Modules Available to Lab Members

Utilities Monitoring

Utilities required to operate equipment are shown in the pop-up menu of each tool on the Equipment page of the Mercury Client. Fig. 12 is an example.



Staff: olek, Project: PR-STAFF

Dependencies: aln2

RESOURCE	UTILITY	TYPE	DESCRIPTION	LOCATION	STA...
aln2	cda	Gas	Compressed Dry Air	280	
aln2	pn2	Gas	Process Nitrogen	266	
aln2	po2	Gas	Process Oxygen	266	
aln2	par	Gas	Process Argon	266	
aln2	c-ar	Gas	Argon Cylinder Gas	380	
aln2	vacuum-pump	Gas	Vacuum Pumps for Equipment	tool	
aln2	pcw	Water	Process Cooling Water	280	
aln2	ac208	Power	AC Power Voltage	180	
aln2	exhaust	Fan	Fume Exhaust	780	
aln2	computer	Misc	Computers associated with equipment	tool	

Fig. 12. Example of the Dependencies list of tystar10, poly-Si LPCVD furnace.

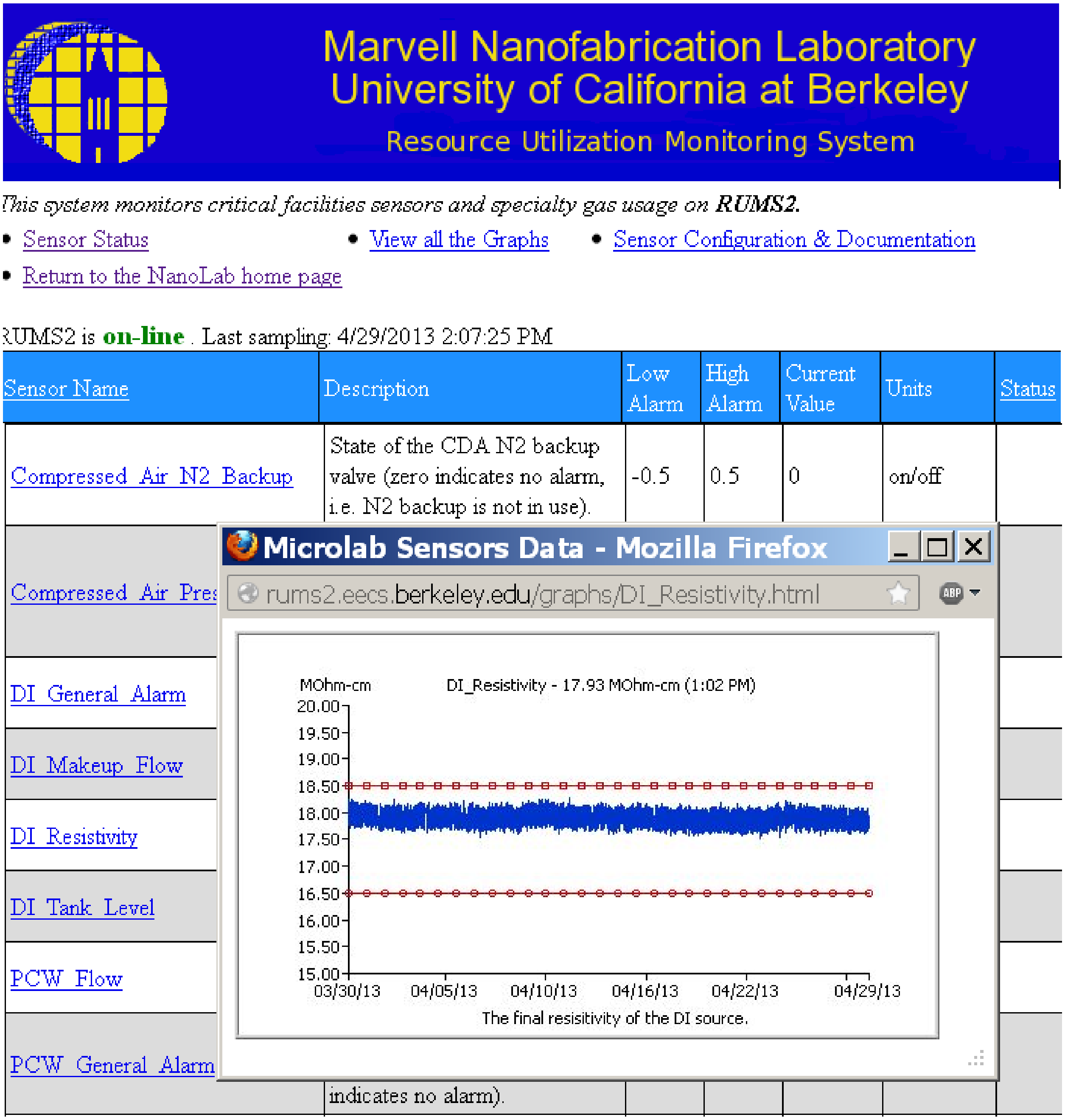
If a project is especially sensitive to environmental conditions, utilities performance can be viewed directly on the NanoLab's web portal (Fig. 9), under Facilities. Clicking on RUMS-Nano will result in a listing shown in Fig. 13. Clicking on a Sensor Name will create a pop-up window with the line chart of the measured data, as shown in the inset in Fig. 13. This program is called RUMS, Resource Utilization Monitoring System and uses its own SQL Server database [3].

Laboratory Fees

The Marvell NanoLab is financed by charging lab-use fees to participating Principal Investigators (PIs). Each student who conducts her/his research in the lab receives a unique account number to which charges are accumulated and billed to the assigned research grant. If there are more than one projects/grants then the user selects the appropriate title to which the activity is posted. The lab member can view his up-to-date lab charges for the month in Mercury Web. (Invoicing is done monthly.)

Lab Manuals

Up-to-date equipment operations manuals are available for viewing, both inside and outside the lab. Mercury Client provides the lab manual in the pop-up menu for the specific tool. Alternately, the complete manual set is available in Mercury Web and through the Marvell NanoLab's web portal, http://NanoLab.berkeley.edu/labmanual.



Marvell Nanofabrication Laboratory
University of California at Berkeley
Resource Utilization Monitoring System

This system monitors critical facilities sensors and specialty gas usage on ***RUMS2.***

- Sensor Status
- View all the Graphs
- Sensor Configuration & Documentation
- Return to the NanoLab home page

RUMS2 is **on-line** . Last sampling: 4/29/2013 2:07:25 PM

Sensor Name	Description	Low Alarm	High Alarm	Current Value	Units	Status
Compressed_Air_N2_Backup	State of the CDA N2 backup valve (zero indicates no alarm, i.e. N2 backup is not in use).	-0.5	0.5	0	on/off	
Compressed_Air_Pres						
DI_General_Alarm						
DI_Makeup_Flow						
DI_Resistivity						
DI_Tank_Level						
PCW_Flow						
PCW_General_Alarm	indicates no alarm).					

Microlab Sensors Data - Mozilla Firefox

rums2.eecs.berkeley.edu/graphs/DI_Resistivity.html

Fig. 13. RUMS sensor status display of NanoLab utilities (partial view).
Inset: Line chart of the measured data with spec limits, for DI water resistivity.

III. Database Design

A clever database design that fits the environment and the requirements of laboratory operations is essential for developing a user-friendly computer management system. The Mercury project started out with a database design shown in Fig. 14. Blocks of the same color indicate the following groups:

🟦 Membership, with attributes of members ID, photo, status, research, advisor, projects; also recognitions, suspensions, and suggestions.
🟩 Grouping, with resource groups, member groups, lineages, groups, and objects.
⬜ All other functions needed for operation of the lab, with parameters for resource used, problem reports, reports history, reserve, calendar, on-line tests, qualify, history.
🟥 Accounting, with project members, charge classes, charge rules, project funds, departments, funds, accounting types, accounting rules, activity, journal rules, journal, ledger, accounting period, and report parameters.
🟧 Resources, with name, locations, equipment, utilities, dependencies, and inv. items.
🟪 Purchase, with items, orders with details, flex-fields, forms, and vendors.

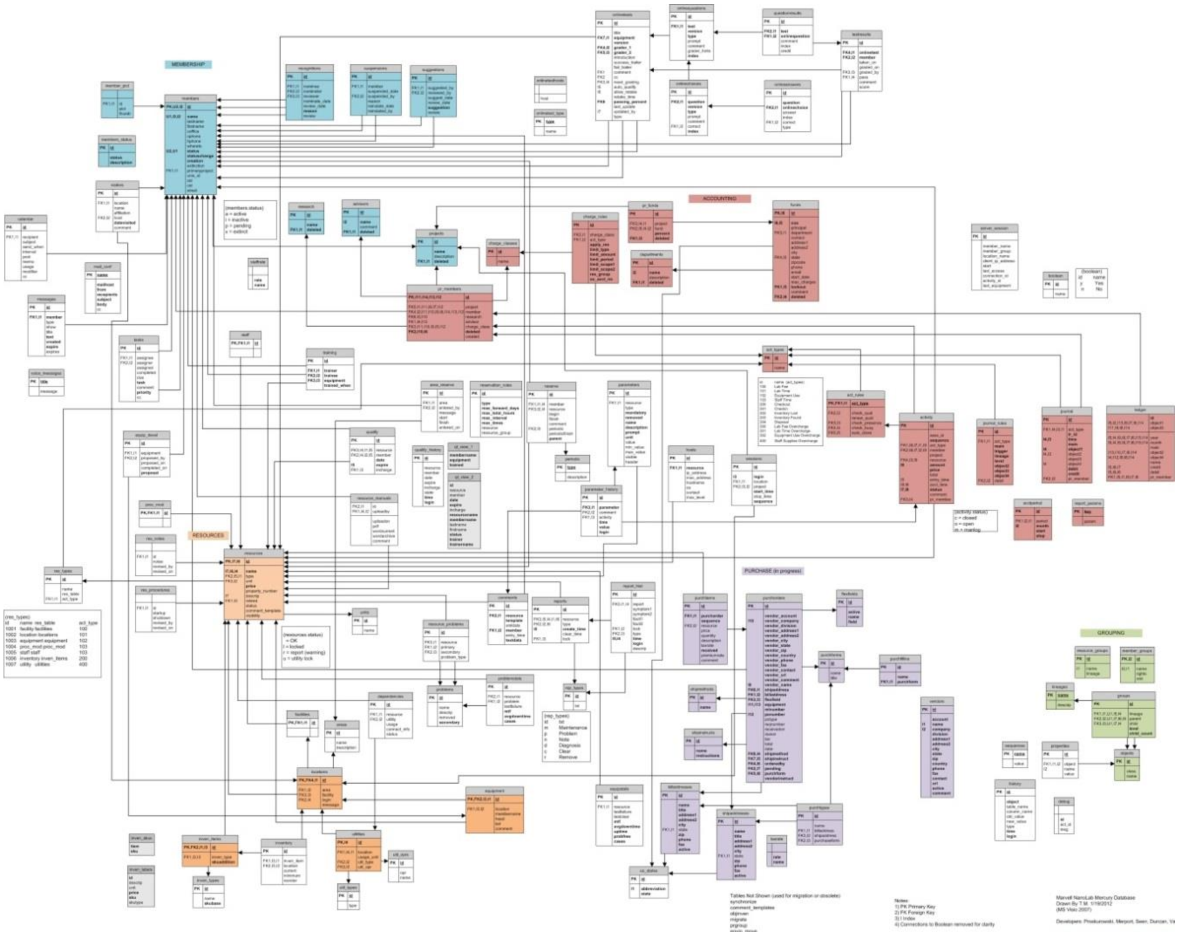


Fig. 14. Mercury database schematic layout (Todd Merport, 2012).

This is a typical relational database design with the data organized in relationships to each other; relationships are specified by column constraints built into the table definitions, transactions are managed by rules and data is manipulated using stored procedures. Mercury is built on the Ingres relational database management system (RDBMS), originally developed at UC Berkeley in the 1970s, now a commercially supported product.

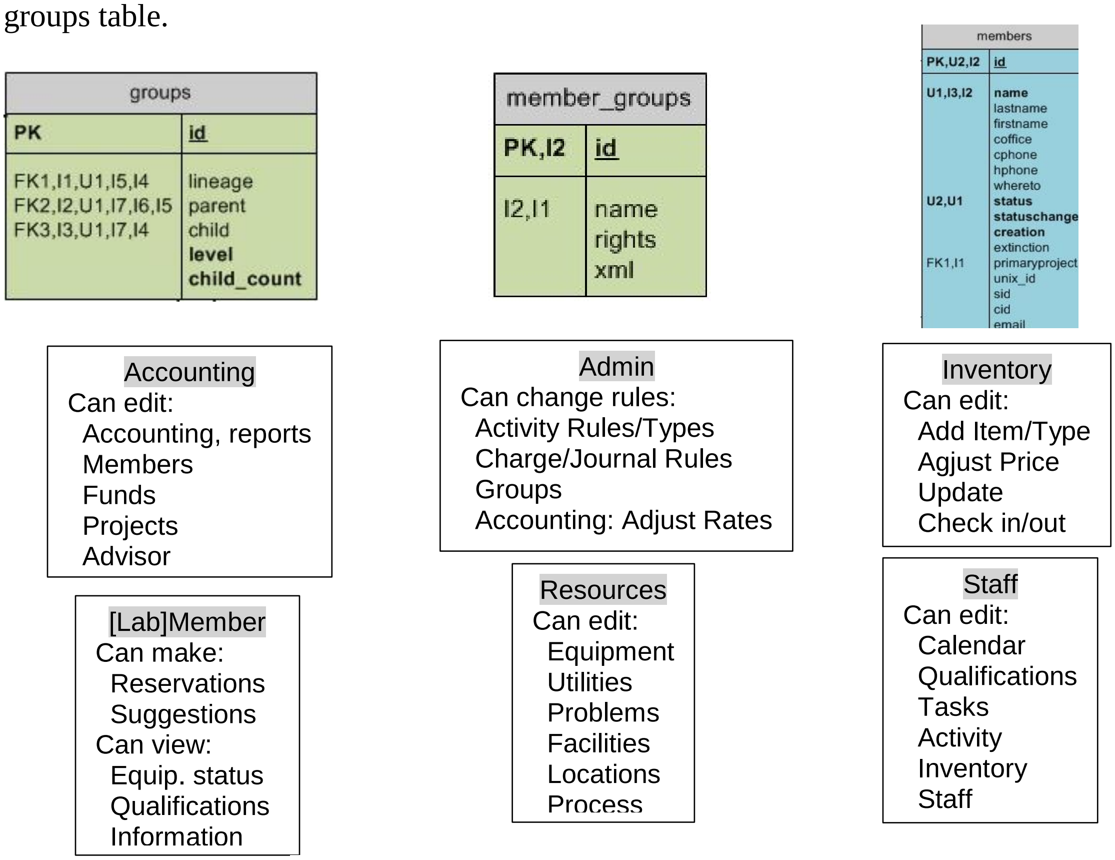
The Mercury database was designed with both relational database and object oriented design patterns. The object patterns provide inheritance such as equipment "is a" resource and polymorphism where table rows are interchangeable objects that can be passed to procedures, grouped, or queried as needs arise. Relational systems provide a high degree of organization, data integrity, standards, and maturity. The goal is a fast, reliable, and flexible system.

Functional Parts

Objects, Groups, and Properties

Mercury has several ways to give things or objects relationships. A relationship can be based on a group of equipment, members, equipment properties, privileges, or journal entry. Mercury provides a way of grouping things in the database by defining them as objects. Objects are built into relationships through the groups, properties, lineages, and objects tables.

The groups table holds a tree like structure. A group's lineage is at the top level. It defines the function of the group. For example the lineage member_groups in Fig. 15 can hold members of a group (identified in the members table) as a parent and the ID of the member is a child in the groups table.



groups	
PK	id
FK1,I1,U1,I5,I4	lineage
FK2,I2,U1,I7,I6,I5	parent
FK3,I3,U1,I7,I4	child
	level
	child_count

member_groups	
PK,I2	id
I2,I1	name
	rights
	xml

members	
PK,U2,I2	id
U1,I3,I2	**name**
	lastname
	firstname
	coffice
	cphone
	hphone
	whereto
U2,U1	**status**
	statuschange
	creation
	extinction
FK1,I1	primaryproject
	unix_id
	sid
	cid
	email

Accounting
Can edit:
Accounting, reports
Members
Funds
Projects
Advisor

Admin
Can change rules:
Activity Rules/Types
Charge/Journal Rules
Groups
Accounting: Adjust Rates

Inventory
Can edit:
Add Item/Type
Agjust Price
Update
Check in/out

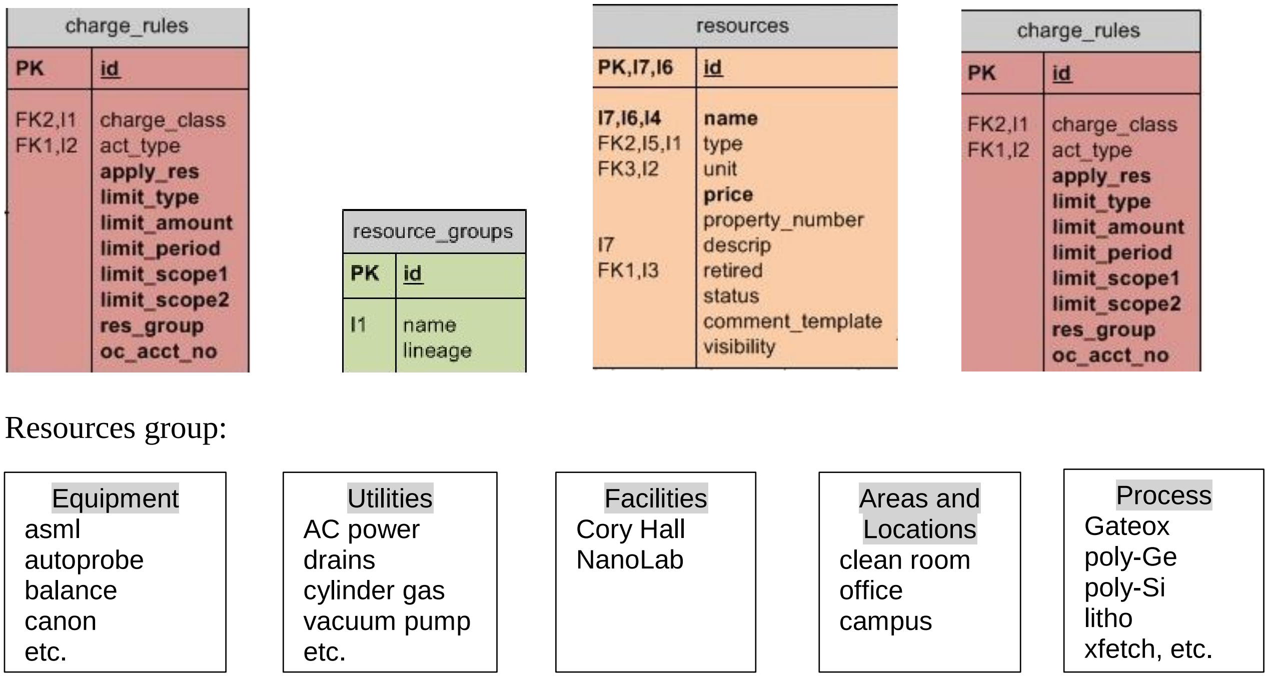
[Lab]Member
Can make:
Reservations
Suggestions
Can view:
Equip. status
Qualifications
Information

Resources
Can edit:
Equipment
Utilities
Problems
Facilities
Locations
Process

Staff
Can edit:
Calendar
Qualifications
Tasks
Activity
Inventory
Staff

Fig. 15.   Example of a group in the database.

Charge_rules in the Accounting module uses a lineage resources (Fig. 16). The parent is the resource group specified in charge rules. The child is the resource used in the current activity. Fig. 17 shows details of the resource 'equipment'.



charge_rules	
PK	id
FK2,I1 FK1,I2	charge_class act_type **apply_res** **limit_type** **limit_amount** **limit_period** **limit_scope1** **limit_scope2** **res_group** **oc_acct_no**

resource_groups	
PK	id
I1	name lineage

resources	
PK,I7,I6	id
I7,I6,I4 FK2,I5,I1 FK3,I2 I7 FK1,I3	**name** type unit **price** property_number descrip retired status comment_template visibility

charge_rules	
PK	id
FK2,I1 FK1,I2	charge_class act_type **apply_res** **limit_type** **limit_amount** **limit_period** **limit_scope1** **limit_scope2** **res_group** **oc_acct_no**

Resources group:

Equipment
asml
autoprobe
balance
canon
etc.

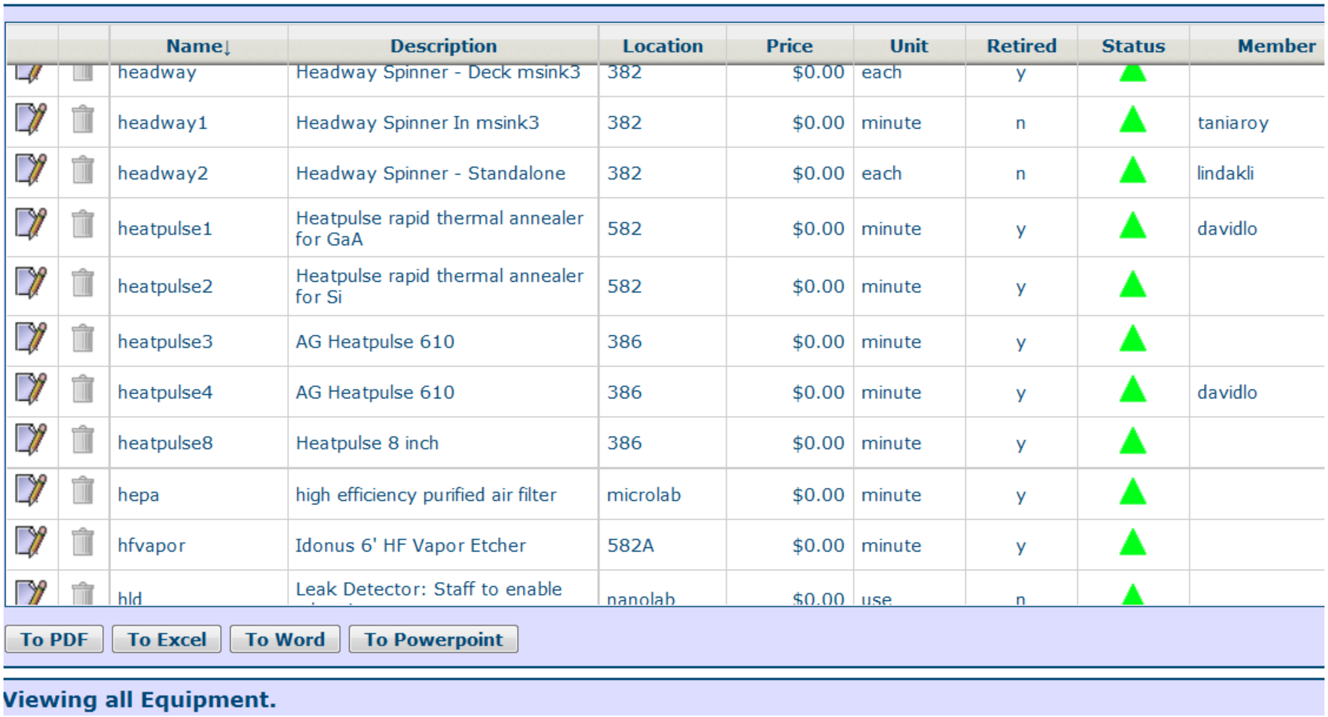
Utilities
AC power
drains
cylinder gas
vacuum pump
etc.

Facilities
Cory Hall
NanoLab

Areas and Locations
clean room
office
campus

Process
Gateox
poly-Ge
poly-Si
litho
xfetch, etc.

Fig. 16. Example of the *resources* group in the database.



		Name↓	Description	Location	Price	Unit	Retired	Status	Member
		headway	Headway Spinner - Deck msink3	382	$0.00	each	y	▲	
		headway1	Headway Spinner In msink3	382	$0.00	minute	n	▲	taniaroy
		headway2	Headway Spinner - Standalone	382	$0.00	each	n	▲	lindakli
		heatpulse1	Heatpulse rapid thermal annealer for GaA	582	$0.00	minute	y	▲	davidlo
		heatpulse2	Heatpulse rapid thermal annealer for Si	582	$0.00	minute	y	▲	
		heatpulse3	AG Heatpulse 610	386	$0.00	minute	y	▲	
		heatpulse4	AG Heatpulse 610	386	$0.00	minute	y	▲	davidlo
		heatpulse8	Heatpulse 8 inch	386	$0.00	minute	y	▲	
		hepa	high efficiency purified air filter	microlab	$0.00	minute	y	▲	
		hfvapor	Idonus 6' HF Vapor Etcher	582A	$0.00	minute	y	▲	
		hld	Leak Detector: Staff to enable	nanolab	$0.00	use	n	▲	

Viewing all Equipment.

Fig. 17. Detail of the resource 'equipment' in the database.

Membership

Membership is comprised of a series of relationships besides just personal data. (Fig. 18.) Each member has one or more advisors, projects and a primary project. Projects have one or more sources of funding, (pr_funds). A member is part of a basic group: admin, staff, [lab]member and database privileges are managed by the Ingres system. Groups define access rights to forms and areas in the Mercury Client or Mercury Web program. A member has a members_status to indicate if he is 'active', 'inactive', or 'extinct'. An 'active' status indicates the member is actively using the facility; an 'inactive' status is a suspended account – the member plans to use the laboratory in the near future; an 'extinct' account indicates the member has left the lab on a permanent basis ─ computer accounts are archived and equipment operation qualifications purged.

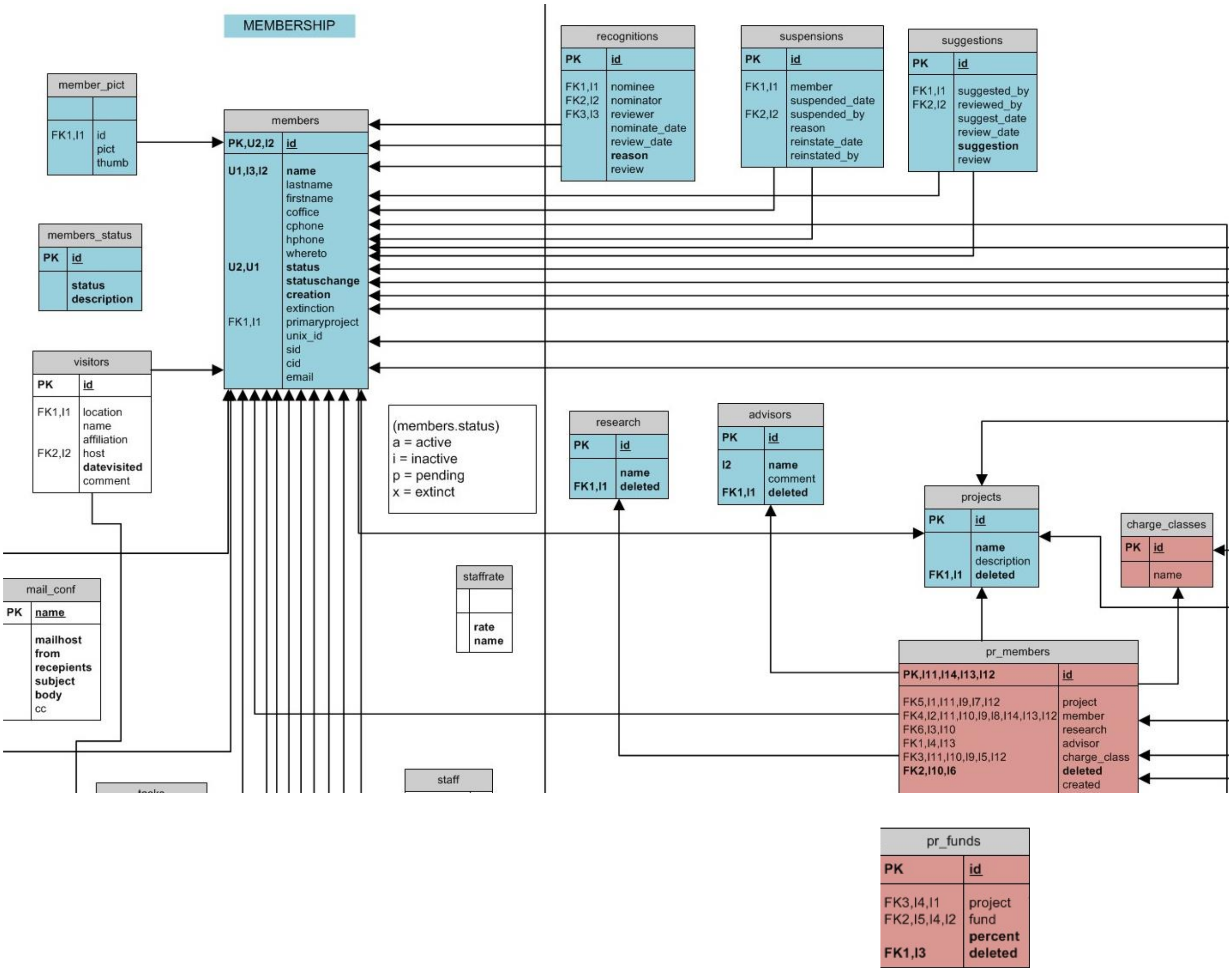


Fig. 18. Membership details of the database.

Resources

A laboratory has resources, such as location, equipment, utilities, and inventory. Resources have a name, description, cost, and status. The resource's status can indicate if a problem has been filed on the resource. (Fig. 19.)

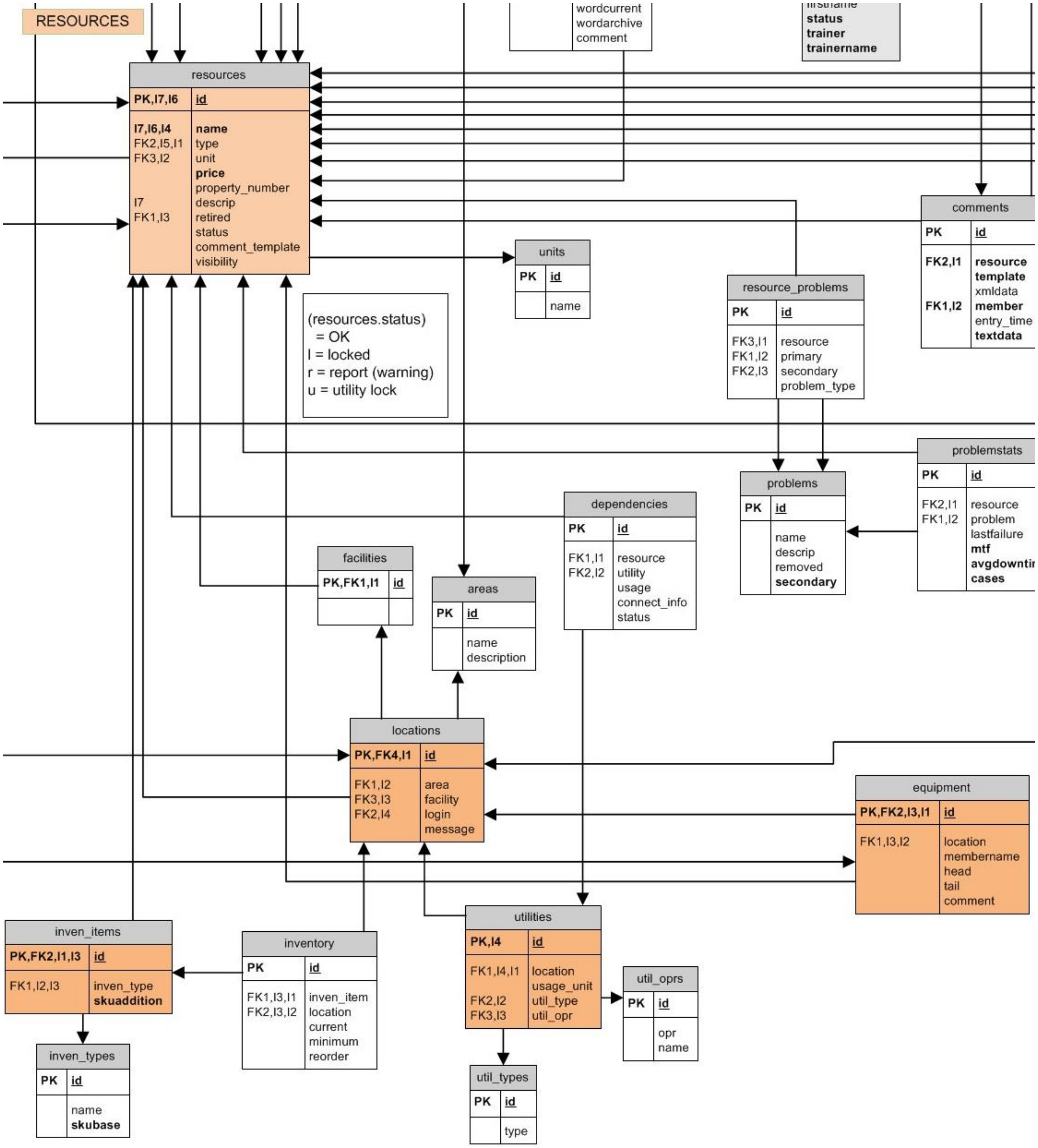


Fig. 19. Schematic of Resources in the database.

The following are defined as resources and inherit all resource attributes. Resources have additional attributes specific to their use in the laboratory.

- ***Equipment:*** a piece of equipment used in the lab. If the equipment is enabled, a member name is associated with it. Equipment also has a head (enable message), tail (disable message).
- ***Utilities:*** a utility is gas, power, or some other item connected to equipment to enable its operation. A utility has a type, location, usage, unit, and operator. Types include power, water, and gas.
- ***Inventory:*** Inventory items have a location, type, current count, minimum count, and re-order amount.
- ***Locations:*** Locations have an area (group of locations), facility (accounting unit), login flag, and message. A location is generally a room with one or more pieces of equipment. Labtime is charged to the location "Marvell Lab" or "anylab" (if more than one locations are managed).
- ***Facilities:*** Each location has a 'facility' A facility represents an accounting unit. An example of a facility would be ANYLAB, MARVELL LAB. Caps are used as a rule. Facilities are listed in the facilities table. A facility is also a resource so monthly access fees can be applied.

Activities and the Accounting Process

Activities are tracked in the lab on the basis on who did what, when. The what is designated as a 'resource' and kept in the resources table. As defined above, resources are equipment, utilities, inventory items, locations, and facilities. Every resource has a type, price, unit, and status (indicating if a problem has been filed. See Fig. 17.)

Any activity or transaction requires recording the following data in the activity table of the Accounting module: member, project, location, activity type, and start/stop time stamps; once an activity is closed, the amount, price, and totals are recorded in the same record. (Figs. 20 and 21.)

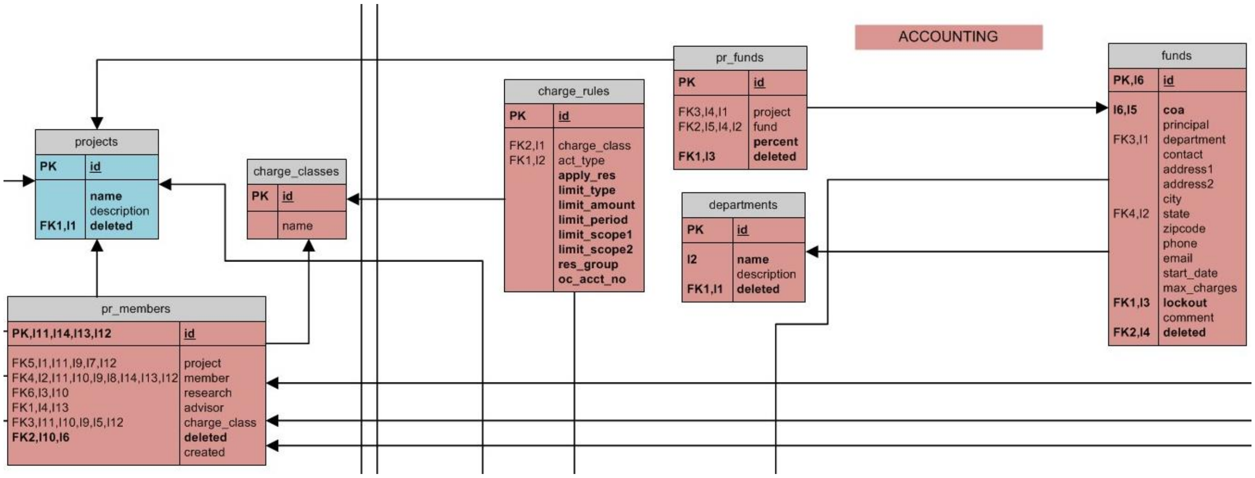


Fig. 20. Schematic of the Accounting module (partial).

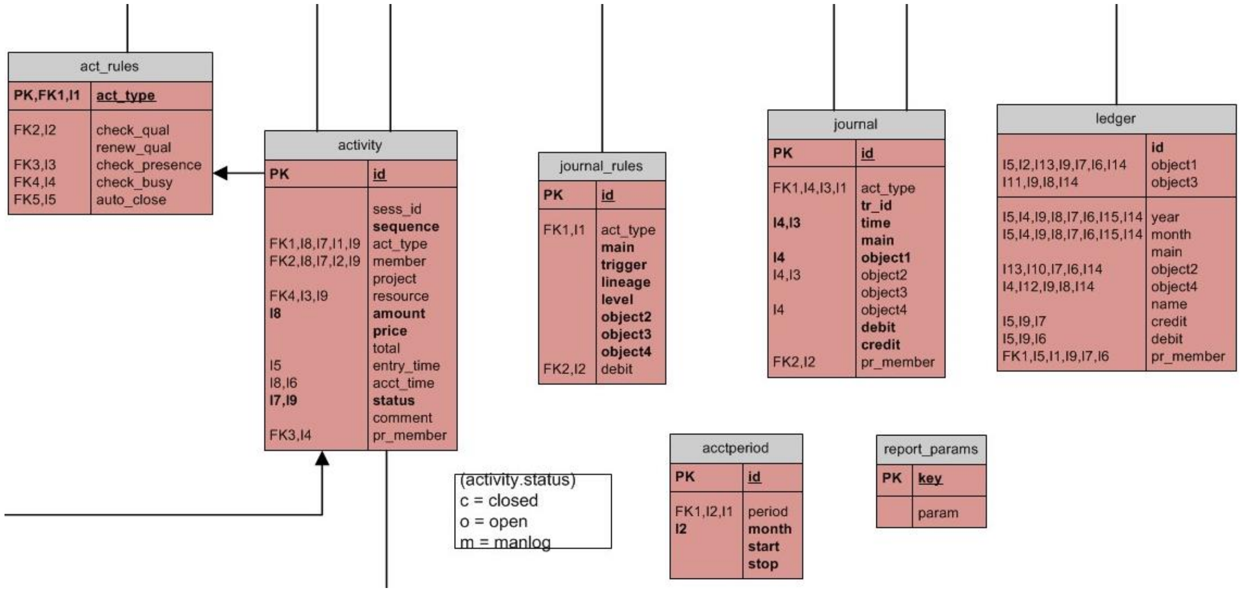


Fig. 21. Schematic of the Accounting module showing the recording of charges.

Members start an activity when they login to the laboratory, enable a piece of equipment, or check in or check out an inventory item. Prior to an activity being recorded it is checked against a series of act_ rules, which check, for example, if a member is qualified for the activity. If the activity passes muster with the rules, a record is placed in the activity table. The activity is marked as open. When the activity is completed it is marked as closed and a series of tests are applied to the activity. Journal entries are created based on the activity type, member, location, and resource. Then charge rules are applied and an additional journal entry may be added as a credit or debit. When the journal entry is complete, a ledger entry is created unique to the member, project, and location. The ledger is a summary of the journal activities based on the accounting period. (Fig. 22.)

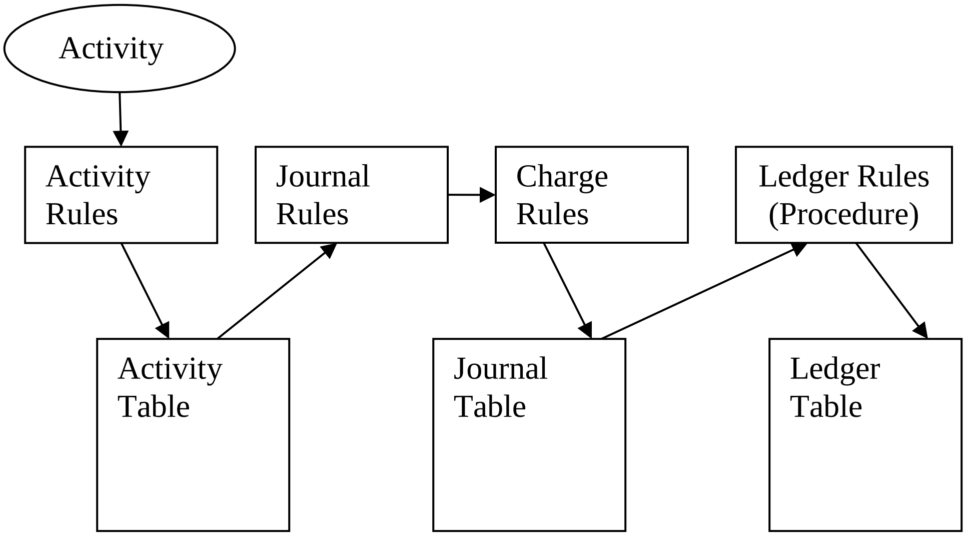
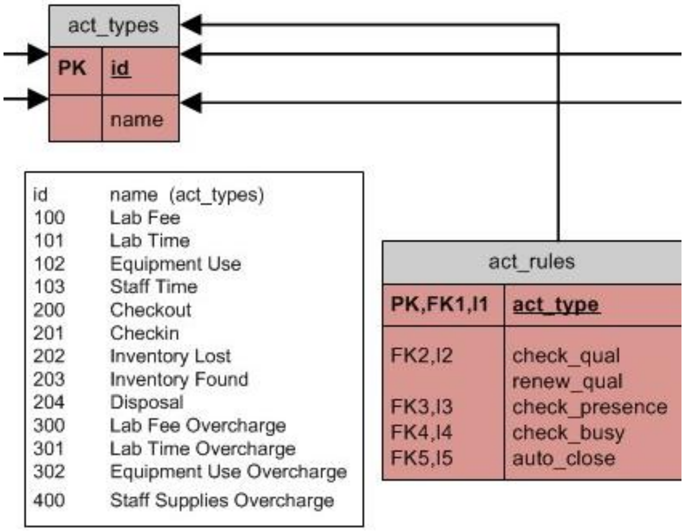


Fig. 22. Accounting activity schematic.

Types of activities accounted for by Mercury are predefined in the activity types (act_types) table. Any entry into the activity table must include its type. The activity rules table (act_rules) is used to validate the activity prior to insertion. One example is shown in Fig. 23: activity type 102 equipment usage requires qualification, extends the qualification for 6 months, tests if the member is in the lab and if the equipment is busy, the auto_close flag is no; after insert another update is required to close the activity (enable, disable).



id	name (act_types)
100	Lab Fee
101	Lab Time
102	Equipment Use
103	Staff Time
200	Checkout
201	Checkin
202	Inventory Lost
203	Inventory Found
204	Disposal
300	Lab Fee Overcharge
301	Lab Time Overcharge
302	Equipment Use Overcharge
400	Staff Supplies Overcharge

Fig. 23. Activity types and rules.

Sessions – A Group of Laboratory Activities

When a member signs into the lab (using Mercury Client), a session is automatically created. (Fig. 24.) All activities are grouped into a session for the interval between sign-in and sign-out. Grouping activities into sessions and giving each activity a sequence number in the session facilitates running the laboratory. For example: a session would be enabling/disabling multiple equipment and resource-use charges, all to be posted to a project. A session includes all chargeable activities. (A session is not created, i.e. no charges are generated, if a member uses Mercury Web.)

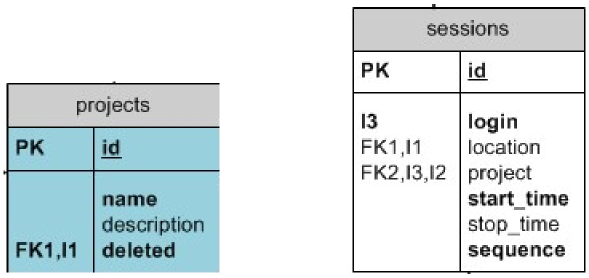


Fig. 24. Sessions of chargeable activities are posted to projects.

Problem Reporting

Problem reports are generated by staff, members, or automatically via a maintenance calendar. The main set of tables involved is shown in Fig. 25. Each report is based on a problem with a resource (resources). A typical case might be a member in the clean room reporting a problem on “oxford” (equipment). The member will select the generalized type of problem out of a pool of pre-defined problems. The problems for a resource are linked via the resource_problems table. Two tables receive inserts when a problem is filed: reports and report_hist. The report_hist table is updated as the resource is evaluated and repaired, the final update will change the status to “Clear” and the clear_time field in reports is set to the current date (See Reference [4]). Email is generated to members with KEYOP or ENG1, ENG2 status in the equipment property tables and qualified members (when the problem is cleared). If the resource is not locked out, i.e. it is still usable but with some limitation, Mercury Web or Mercury Clent will show the resource with a semaphore that is yellow (a warning). A severe problem gets the red semaphore – the member is unable to enable the equipment. This logic is checked with the activity_insert_proc stored procedure. If a problem report is filed on a utility, all dependent equipment can be locked out. For example, a liquid nitrogen (LN) problem can lockout all equipment dependent on LN. The semaphore seen by the member in this case is black. Problem reports can be generated on equipment, locations, and utilities.

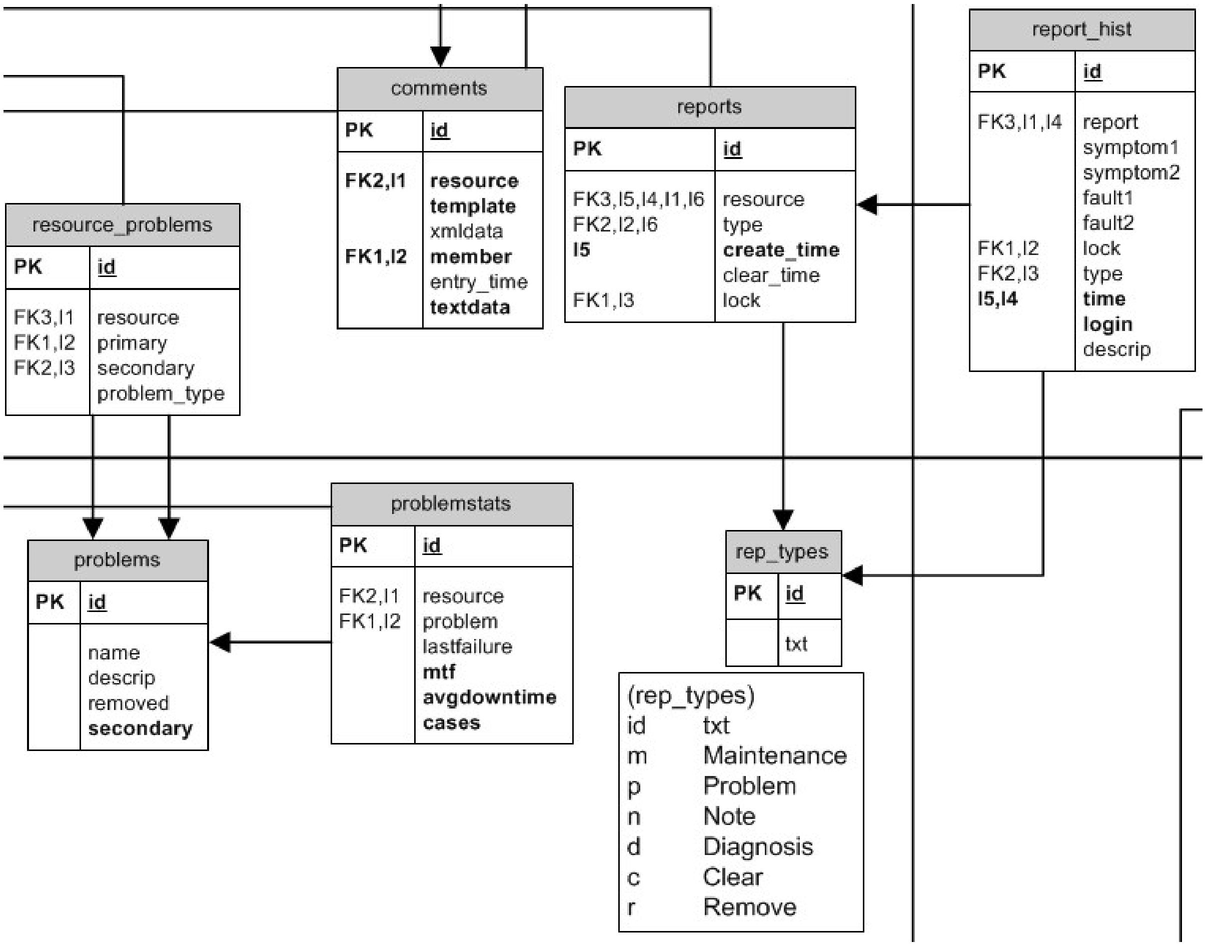


Fig. 25. Problem reporting details of the database.

Reservations

Members make reservations on resources (equipment) through Mercury Web. To prevent a free for all on popular equipment, various rules are imposed on equipment groups or an individual machine. Rules are held in the reservation_rules table Fig. 26.) In a given period, reservations are limited by number of times, total time, and maximum time for a single instance. The member that made the reservation receives a reminder notice early in the morning of the reserved day. Enforcing reservations is a matter of lab policy. Members attempting to enable equipment that has an upcoming reservation will be warned with a pop-up message on Mercury Client.

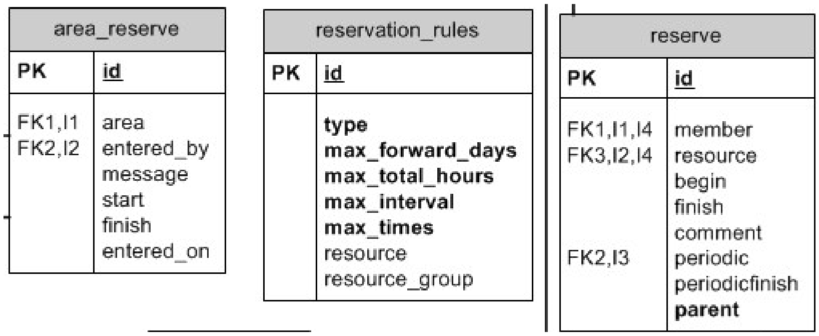


Fig. 26. Equipment reservation section.

Qualifications

Members get qualified on equipment through training and testing. Once qualified, they get on the email alias for the equipment. Each time a member uses the equipment, the qualification is extended for six months. When the qualification expiration date is near, members will receive a reminder email to extend their qualification. The trainer of the qualified member gets noted in the training table so management can recognize good citizenship. (Fig. 27.)

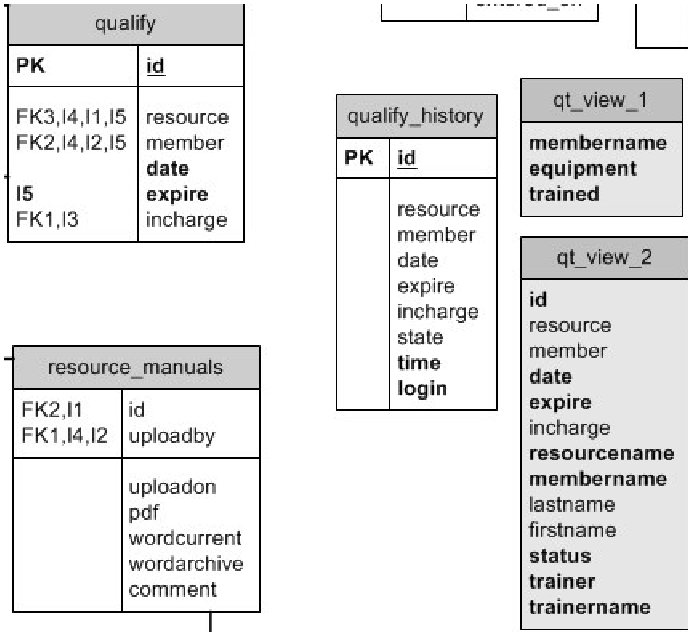


Fig. 27. Qualifications section.

On-Line Tests

Members can take tests as part of the qualification process to use the NanoLab and individual equipment using the On-Line Tests feature of Mercury Web. Fig. 28 shows the tables involved. Essentially, a test has questions, answers, choices, and a result. Designated staff design and grade tests.

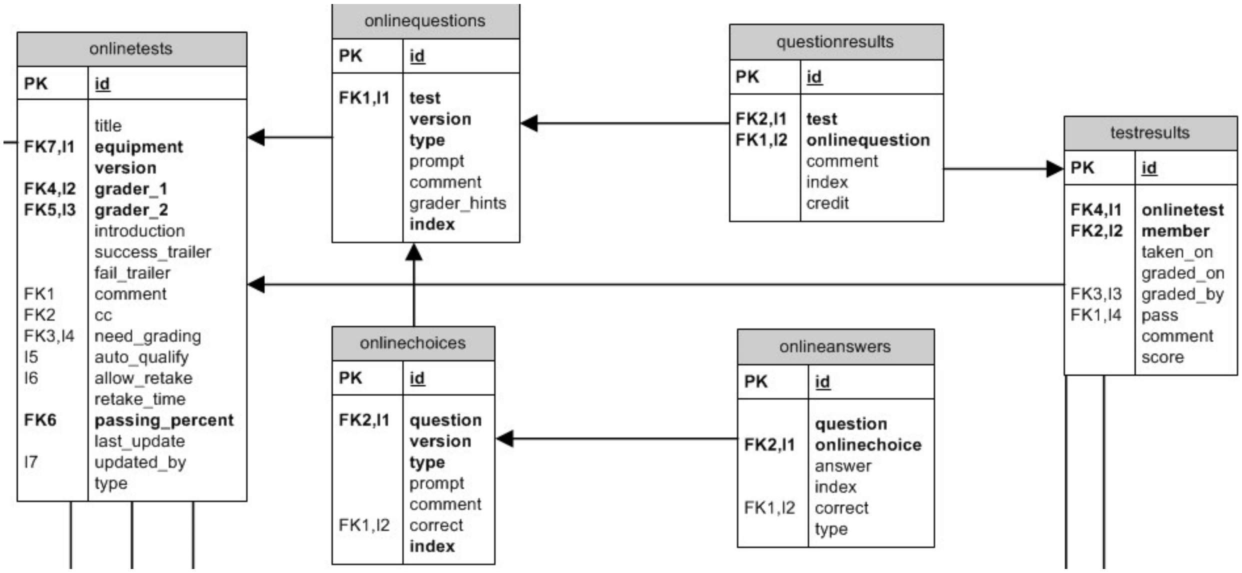


Fig. 28. On-line tests.

Inventory

Inventory is another resource which is recorded in a Session (and billed at the end of the session.) Specialty gas use by an equipment is calculated automatically from input data from RUMS [3] and recorded in the session by Mercury Client. Inventory is managed through Mercury Web.

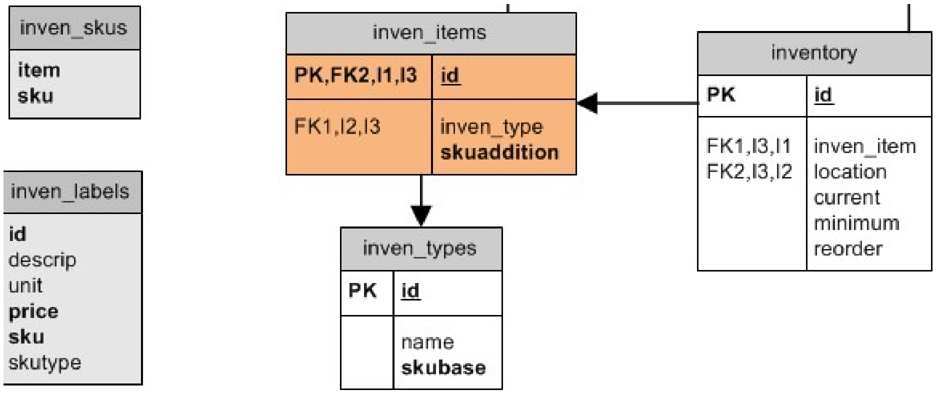


Fig. 29. Inventory details in the database.

IV. Database Details

Most of the logic is implemented in SQL stored procedures. (Data is inserted into tables by stored procedures.) Following is a description of the accounting process.

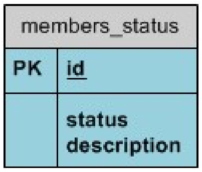
Tables with Pre-Defined Data

Data in these tables can be changed only with the right privileges.

Members Status (members_status)

ID	Status	Description
1219	a	Active
1220	i	Inactive
1221	x	Extinct
1222	p	Pending

Columns:
ID, row id
Status, character indicating member status
Description of status



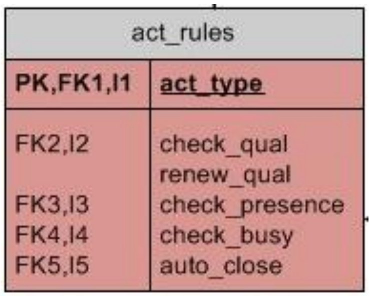
Activity Types (act_types)

ID	Name
100	Lab Fee
101	Lab Time
102	Equipment Use
103	Staff Time
200	Checkout
201	Checkin
202	Inventory Lost
203	Inventory Found
204	Disposal
300	Lab Fee Overcharge
301	Lab Time Overcharge
302	Equipment Use Overcharge
400	Utilities

Columns:
ID, predefined ID
Name, name of type

Activity Rules (act_rules)

act_type	check_qual	renew_qual	check_presence	check_busy	auto_close
100	n	NULL	n	n	y
101	y	NULL	n	n	n
102	y	6 months	y	y	n
103	n	NULL	n	n	y
300	n	NULL	n	n	y
301	n	NULL	n	n	y
302	n	NULL	n	n	y
400	n	NULL	n	n	y



act_rules

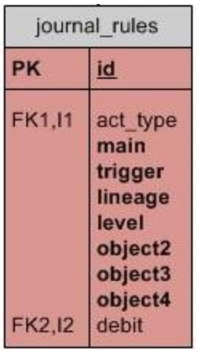
PK,FK1,I1	act_type
FK2,I2	check_qual
	renew_qual
FK3,I3	check_presence
FK4,I4	check_busy
FK5,I5	auto_close

Columns:
act_type – id from act_types table
check_qual -- check qualifcations for this activity
renew_qual -- renew qualifications (for 6 months)
check_presence -- require member to be signed in for this activity
check_busy -- make sure there is no contention with another activity for this resource
auto_close -- close this activity right away (for manual log).

Notes:
activity_insert_rule after INSERT on activity -> activity_insert_proc
activity_update_rule after UPDATE on activity -> activity_close_proc
activity_close_proc called when activity is updated. Calls activity_journal_proc and activity_oc_proc.
activity_journal_proc gets information from closed activity, make journal entry, and uses journal_rules.
activity_oc_proc applies charge rules.

Journal Rules (journal_rules)

act_type	trigger	object2	object3	object4	debit
100	facility	null	null	null	n
100	member	facility	project	act_type	y
101	resource	member	res_group	project	n
101	member	facility	project	act_type	y
102	resource	member	res_group	project	n
102	member	facility	project	act_type	y
103	resource	member	res_group	null	n
103	member	facility	project	act_type	y
301	resource	member	res_group	project	y
301	member	facility	project	act_type	n
302	resource	member	res_group	project	y
302	member	facility	project	act_type	n
300	resource	member	res_group	null	y
300	member	facility	project	act_type	n



journal_rules	
PK	id
FK1,I1	act_type **main** **trigger** **lineage** **level** **object2** **object3** **object4**
FK2,I2	debit

Columns:
Activity Type (act_type). Activity types are based on real activities such as lab usage, equipment usage, and inventory check outs.

Main (main).

Trigger (trigger). Each trigger for a given activity will create a separate line in the journal. The trigger resource is listed as object1 in the journal. Usually a given activity will create two entries in the journal. A line indicating a credit to the resource used, and a line debiting a member, facility, and project.

Lineage (lineage), setting object1 by looking at parent group
Level (level), negative number – home many levels to backtrack to find group
Object2 , a resource
Object3 , a resource
Object4, a resource
Debit, create a debit or credit for this entry.

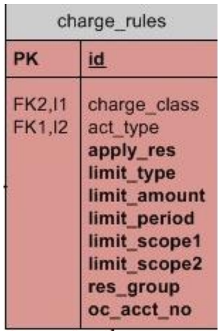
Charge Classes (charge_classes)

ID	Name
800	no access
801	member
802	staff
803	bmla
804	professor

Columns:
ID – predefined ID for class
Name – class name

Charge Rules (charge_rules)

charge_class	act_type	apply_res	limit_type	limit_amount	limit_period	limit_scope1	limit_scope2	res_group
801	100	1067	a	1.00	m	m	r	0
801	101	1072	t	1200.00	m	m	r	0
801	102	1067	t	1400.00	m	m	a	0



charge_rules	
PK	id
FK2,I1 FK1,I2	charge_class act_type **apply_res** **limit_type** **limit_amount** **limit_period** **limit_scope1** **limit_scope2** **res_group** **oc_acct_no**

Example for charge_class members
(act_type 100 lab fee, 101 labtime, 102 equipment use)
(apply_res 1067 MARVELL LAB as facility, 1068 Marvel Lab as location)

Columns:
charge_class -- each project-member has a charge class that defines charge rules. The charge classes are defined in the table charge_classes.
act_type -- activity types indicate what activity occurred such as equipment usage, lab time, inventory check in. Referenced from table act_types.
apply_res – this could be a room, facility, or a piece of equipment.
limit_type -- defines if the limit should apply to an amount ‘a’ or total ‘t’ from a group of act.
limit_amount -- cut -off amount for rule. For example equipment might be $1400 and labtime $1200. For charge_class of staff use $0. Single activities use $1.
limit_period -- the period that this rule applies to: day ‘d’, month ‘m’, or year ‘y’.
limit_scope1-- member based scoping. Valid entry are ‘m’, member ONLY.
limit_scope2 -- resource based scope. Define what resources this rule applies: ‘r’ is is single resource like labfee, ‘a’ is all resources, ‘g’ is a group of resources ‘x’ is except resources indicated in res_group column.
res_group -- a parent object id in the groups table under lineage ‘resource’. If the entry is ‘0’, then use the location. A res_group can be a location. All equipment as children for this location will be included/excluded based on limit_scope2. *If a resource in apply_res is in a group under lineage ‘resource’, the res_group column must be populated with its parent id.*

Charge Rules Example (names are used instead of ids of flags)

charge_class	activity type	resource	limit type	limit amount	limit period	limit scope1	limit scope2	resouce group
bmla	Lab Fee	MARVELL LAB	activity	1	month	member	resource	
bmla	Lab Fee	MICROLAB	activity	1	month	member	resource	
bmla	Lab Time	Marvell Lab	total	1600	month	member	resource	
bmla	Lab Time	microlab	total	1600	month	member	resource	
member	Equipment Use	MARVELL LAB	total	1400	month	member	all	
member	Equipment Use	MICROLAB	total	1400	month	member	except	197 Cory
member	Equipment Use	crestec	total	2400	month	member	resource	197 Cory
member	Lab Fee	MARVELL LAB	activity	1	month	member	resource	
member	Lab Fee	MICROLAB	activity	1	month	member	resource	
member	Lab Time	Marvell Lab	total	1200	month	member	resource	
member	Lab Time	microlab	total	1200	month	member	resource	
professor	Equipment Use	MARVELL LAB	total	1400	month	member	all	
professor	Equipment Use	MICROLAB	total	1400	month	member	all	
professor	Lab Fee	MARVELL LAB	activity	1	month	member	resource	
professor	Lab Fee	MICROLAB	activity	1	month	member	resource	
professor	Lab Time	Marvell Lab	total	1200	month	member	resource	
professor	Lab Time	microlab	total	1200	month	member	resource	
staff	Equipment Use	MARVELL LAB	total	0	month	member	all	
staff	Equipment Use	MICROLAB	total	0	month	member	all	
staff	Lab Fee	MARVELL LAB	activity	0	month	member	resource	
staff	Lab Fee	MICROLAB	activity	0	month	member	resource	
staff	Lab Time	Marvell Lab	total	0	month	member	resource	
staff	Lab Time	microlab	total	0	month	member	resource	

Tracing a Session

Session Creation Sequence

1. Click on Mercury Client.
2. Login screen appears. Database user is “defuser2”
3. Member types in login name. Database queries for last project. Shows up in drop-down.
4. Member types password. New database session is created. The validation of the login is checked with the connect() method (through the jdbc driver). The ip address is checked after authentication for a valid remote host (kept in hosts table). If the remote host is not valid, an Exception is thrown but it is caught by the Server and returned to the client. For the gui client, an error appears as a popup message. If a login corresponds to OPERATOR, the user is authenticated but database macros are not invoked to log the user into the lab. This is useful for utility scripts that access the database. If the remote_location macro is set by the client, the ip address of the originating socket is reset to the macro's value. This is used when invoking a client from a terminal server as a way to determine the real originating ip address; the new ip address still must be associated with a registered host and location. The remote_location macro is registered as a user macro. It is restored if the user logs out and logs in again from the same client invocation.
5. Member now owns session. Three database groups defined and managed by rmdbs: admin, member, and staff.
6. If all is well at this point, the server session (DBSession) will create a labtime session and activity. A row is inserted into the sessions and activity table. The session id is stored on the Mercury Server. The client will access this ID when inserting an equipment usage activity (it must be linked to the current session). A stored procedure in the database will automatically increment the sequence of the session for this activity and link the session id into the activity table.

Session Table – entries created by the Mercury server after authentication.

id	login	location	project	start_time	stop_time	sequence
17916	merport	1072	1274	Sep …	Sep …	1
17917	merc_m01	1072	1275	Sep …	Sep …	1

Journal Example

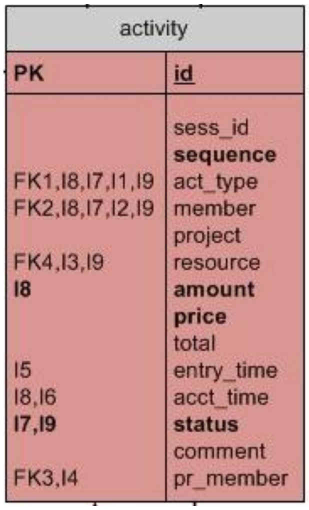
act_type	tr_id	time	object1	object2	object3	object4	debit	credit
101	14	Sep 3 2008 14:32:11	1072 (location)	1216 (member)	0 (null)	1275 (project)	0.00	50.40
101	14	Sep 3 2008 14:32:11	1216 (member)	1067 (Facility)	1275 (project)	101 (act_type)	50.40	0.00

Tables used in Accounting

Activity

id	sess_id	sequence	act_type	member	project	resource	amount	price	total	entry_time	act_time	status
7	17909	2	102	1216	1275	4507	2859.00	0.64	1829.76	entry_time	acct_time	c
8	17914	1	101	1211	1274	1072	0.00	0.00	0.00	entry_time	acct_time	c

Columns:
id – auto generated id
sess_id – from sessions.id
sequence – sequence of this session
act_type – from act_types.id
member – from members.id
project – from projects.id
resource – from resources.id
amount – number of units used in this activity
price – price per unit
total – amount * price
entry_time – time when entry is inserted into activity table (activity opened).
acct_time – time when activity is closed (elapsed time is acct_time – entry_time).
status – o open, c closed, m manlog, R re-create
comment (not shown) – for manual entries.



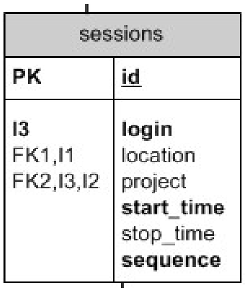
activity	
PK	id
	sess_id
	sequence
FK1,I8,I7,I1,I9	act_type
FK2,I8,I7,I2,I9	member
	project
FK4,I3,I9	resource
I8	**amount**
	price
	total
I5	entry_time
I8,I6	acct_time
I7,I9	**status**
	comment
FK3,I4	pr_member

Sessions

– entries created by the Mercury server after authentication.

id	login	location	project	start_time	stop_time	sequence
17916	merport	1072	1274	Sep …	Sep …	1
17917	merc_m01	1072	1275	Sep …	Sep …	1

Columns:
id – system generated id
login – member that logged into the Mercury client
location – location id
project – member project for this session
start_time – session start time
stop_time – session end time
sequence – number of sequences in this session



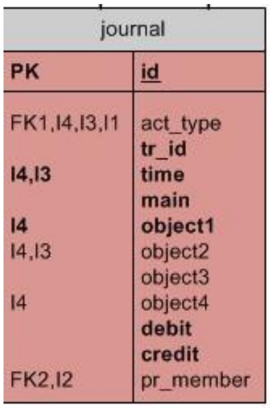
sessions	
PK	id
I3	**login**
FK1,I1	location
FK2,I3,I2	project
	start_time
	stop_time
	sequence

Journal

id	act_type	tr_id	time	main	object1	object2	object3	object4	debit	credit
17918	102	7	acct_time	1	4507	1216	0	1275	0.00	1829.76
17921	102	7	acct_time	1	1216	1067	1275	102	1829.76	0.00
17924	302	12	acct_time	1	4507	1216	0	1275	429.76	0.00

Columns:
id – system generated act_type – references act_types.id
tr_id – references activity.id
time – time activity was closed
main – always 1 (future use for multiple statements?)
object1-4 – see journal rules below.
debit
credit

journal_rules table: used by activity_journal_proc
for journal entries. The trigger column maps to object1 in the journal.

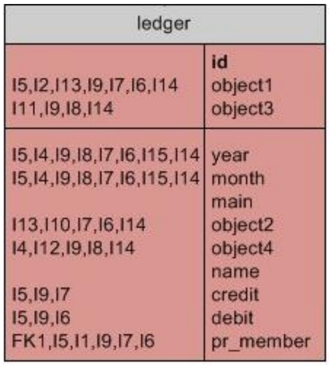


journal	
PK	**id**
FK1,I4,I3,I1	act_type
	tr_id
I4,I3	**time**
	main
I4	**object1**
I4,I3	object2
	object3
I4	object4
	debit
	credit
FK2,I2	pr_member

Ledger

ID	year	month	main	object1	object2	object3	object4	name	debit	credit
17932	2009	0	1	1072	1216	0	1275	NULL	51.03	0.00
17934	2009	3	1	1216	1067	1275	101	NULL	0.00	51.03

Columns:
id – system generated id
year – fiscal year
month – month in fiscal year, month 1 = july. month 0 = summary for entire fiscal year.
main –
object1-4 -- defined from journal
name --
debit – summation of debits for year, month
credit – summation of credits for year, month
ledger is updated in journal_insert_proc. Summary of charges when
object1=object1…object4=object4



ledger	
	id
I5,I2,I13,I9,I7,I6,I14	object1
I11,I9,I8,I14	object3
I5,I4,I9,I8,I7,I6,I15,I14	year
I5,I4,I9,I8,I7,I6,I15,I14	month
	main
I13,I10,I7,I6,I14	object2
I4,I12,I9,I8,I14	object4
	name
I5,I9,I7	credit
I5,I9,I6	debit
FK1,I5,I1,I9,I7,I6	pr_member

Stored Procedures/Triggers (Rules)

Mercury takes advantage of relational database technology: efficiency, data integrity, redundancy, and maturity. Much of the code to insure valid entries, user permissions, calculations, and data entries are implemented by database stored procedures. Keeping this type of code close to the data helps uncouple "business logic" from the client side – where change is continual. This school of thought is widely accepted although debates still rage on.

Two examples of stored procedures in Mercury are given. The activity_insert_procedure is triggered when a member logs in, enables equipment, or does anything that has a charge associated with their action. The activity_close_proc is triggered when the activity has ended such as logging out of the lab or disabling equipment.

Activity Insert Proc (activity_insert_proc)

Description:
Test parameters for activity entry based on activity rules, session values, activity status, and authenticated user.
Usage:
activity_insert_proc (id, sess_id, member, seq, resource, act_type, status, amount)
id, the row id of this activity
sess_id, session id for this group of activities
member, id of the member triggering this activity
seq, sequence number for this session
resource, id of resource that is the object of this activity
act_type, activity type from the act_types table,
status, o-open, c-closed,a-auto-close,m-manlog.
amount, total use (can be minutes,days, qty).
Callers:
activity_insert_rule (after insert on activity). New values are passed to activity_insert_proc.
DBSession.java FormUtilities.java
Return Value:
None

Examples:
labtime insert from dbsession object:

```
INSERT into ACTIVITY(sess_id,act_type, member,resource)
VALUES(:sess_id,101,:userid,:location)
```

Note :sess_id,:userid,:location are variable names

inserting an equipment activity

```
INSERT into activity(act_type,member,sess_id,resource)
VALUES(102,:userid,:sessionid,:resourceid)
```

Note :userid,:sessionid,:resourceid are variable names

Implementation:

1. Test activity type and status. If the activity type is 100, labfee and status not ‘m’ manlog, raise an error (labfees can only be applied with type manlog).
2. Create a unique row id.
3. Find out who has authenticated for the current session (ingres variable current_user). Determine the members’s row id. Check if member belongs to the staff or admin group.
4. Test activity status. If type ‘m’, manlog, member must be staff. Otherwise raise an error.
5. If activity status ‘m’, and member is staff. Close the activity. Otherise
6. determine the sequence, session location, and project from the sessions table.
7. Determine the session area from the location table.
8. Update the sequence number (activity number for this session). Sanity check on sequence number.
9. Make sure non-staff have a valid project (otherwise raise an error).
10. If the activity type is 101, labtime, charge access fee if needed (labfee) and close activity.
11. Start testing activity rules.
    a. see if resource is locked (ie, for a problem report)
    b. see if resource is busy (someone else is using it)
    c. see is the user is in the proper location to create activity (except staff)
    d. test qualification. Execute procedure qualify_proc. This procedure will raise an error if a member is not qualified.
12. Update activity entry time to ‘now’, entry_time.
13. Close activity is amount is greater than 0 or activity status = ‘a’, auto_close.

Activity Close Procedure (activity_close_proc)

Description:

This procedure runs when the status field is updated in the activity table. It is triggered from the activity_close_rule. Nicely closes activity. Sets the activity close time (acct_time) to ‘now’ and sets amount based on the entry time and the close time (unless the unit is each or use).
Determines the charge class and calls Activity Journal Procedure and
Activity OverCharge Procedure.

Usage:

activity_close_proc(id , old_status , status)
id, activity row id,
old_status, previous activity status prior to update.
status, newly updated status.

Caller:

Activity Close Rule (activity_close_rule) when activity status is updated.

Return Value:

none.

Implementation:

1. Test old and new status.
    a. if old.status is ‘R’, recreate and new.status is ‘c’, close, return.
    b. if old_status is ‘c’ and new status is not ‘R’, recreate, return
    c. if new.status is not ‘c’ or ‘R’, raise an error.
2. Get fields from the activity table for this activity.
3. Populate charge class from primary project for the member creating this activity. (project is normally determined from the session information. If the old.status is ‘m’ and new.status is not ‘R’, set the charge_class based on the project in the session for this activity (why not populate directly from activity table?).
4. Populate the location and facility variable for this activity’s resource.
5. Find the userid for the currently authenticated user. Determine if user is staff.
6. If status is not ‘R’
    a. if activity member is not authenticated user, update activity row with a comment (‘activity was closed by ..’).
    b. if the price is zero in the activity table, find the unit and price for this resource.
7. If status in not ‘R’
    a. Determine amount.
    b. update total
    c. update activity with total and activity close time (acct_time).
8. If new.status is not ‘R’ update the equipment.membername and sessions.stoptime.
9. If new status is ‘R’, update set the status in the activity table to the old.status and set a comment ‘Activity Recreated’.
10. Execute procedure activity_journal_proc(id,acct_time).
11. Execute procedure activity_oc_proc(id, charge_class, old.status).

V. Development and Operational Technologies (Dev-Ops)

MercuryServer and MercuryClient Components

MercuryServer and MercuryClient are both Java applications that make up two of the three tiers in the Mercury system (Fig. 1). MercuryServer acts as man-in-the-middle handling communications between the client and the database. MercuryServer is started up on a server type computer and runs in the background listening for connections on a secure socket port.

MercuryClients are started from members' desktops or terminal servers and connect to the server via network socket protocols. Once the client receives a socket, the Server will create a Session thread with a unique session ID (independent of a database session). The MercuryClient has a limited number of rights with the initial connection as an anonymous user. Server side classes are instantiated and added to a list of objects available for the session. Once the member is authenticated, a hosts table will be checked to insure the client is in a valid predefined network or at a specific IP address. An authenticated client is permitted to submit macros to the server that are predefined SQL statements. The client can also invoke remote procedure calls on session objects such as the EquipmentManager to send messages to data acquisition and control systems. All server objects accessible to the client are managed by an Authority Manager. (Figs. 30, 31.)

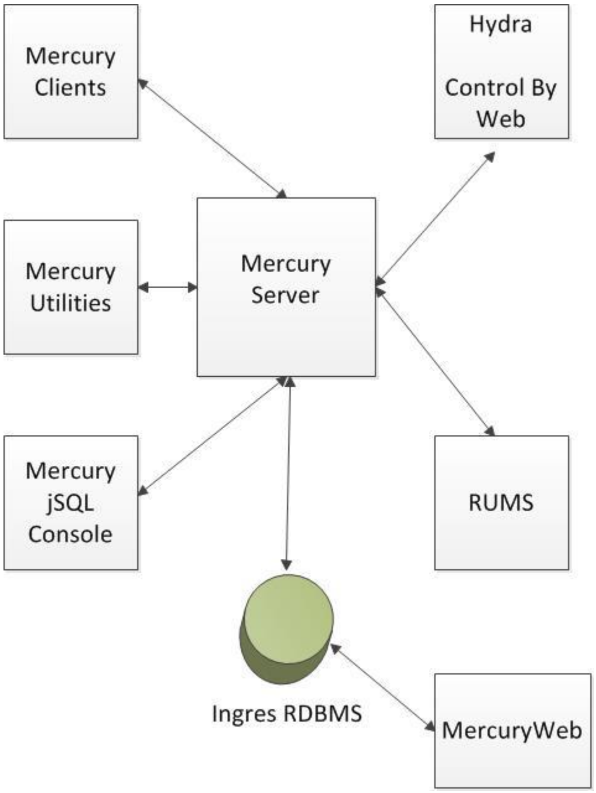


Fig. 30. Mercury System components.

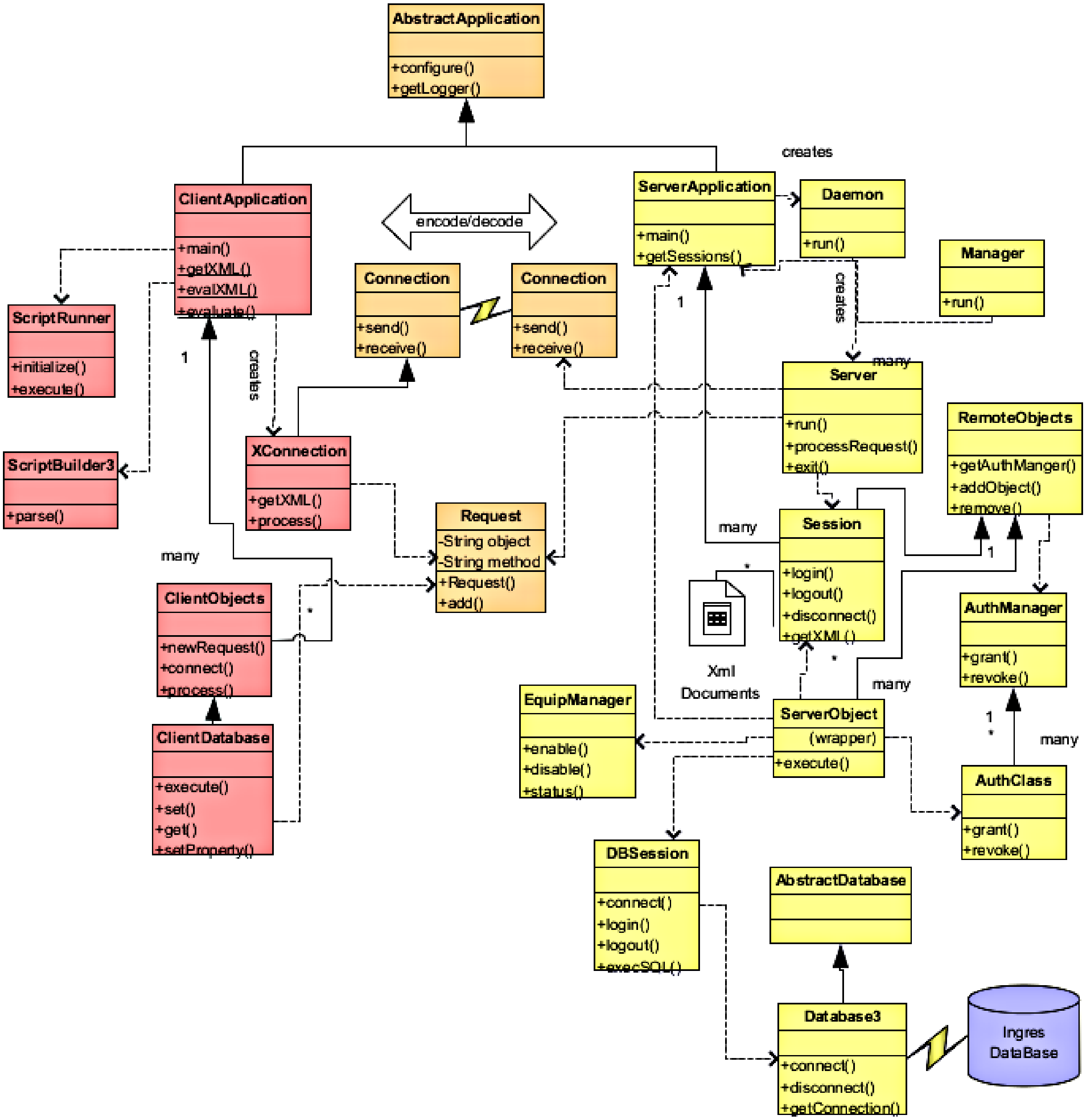


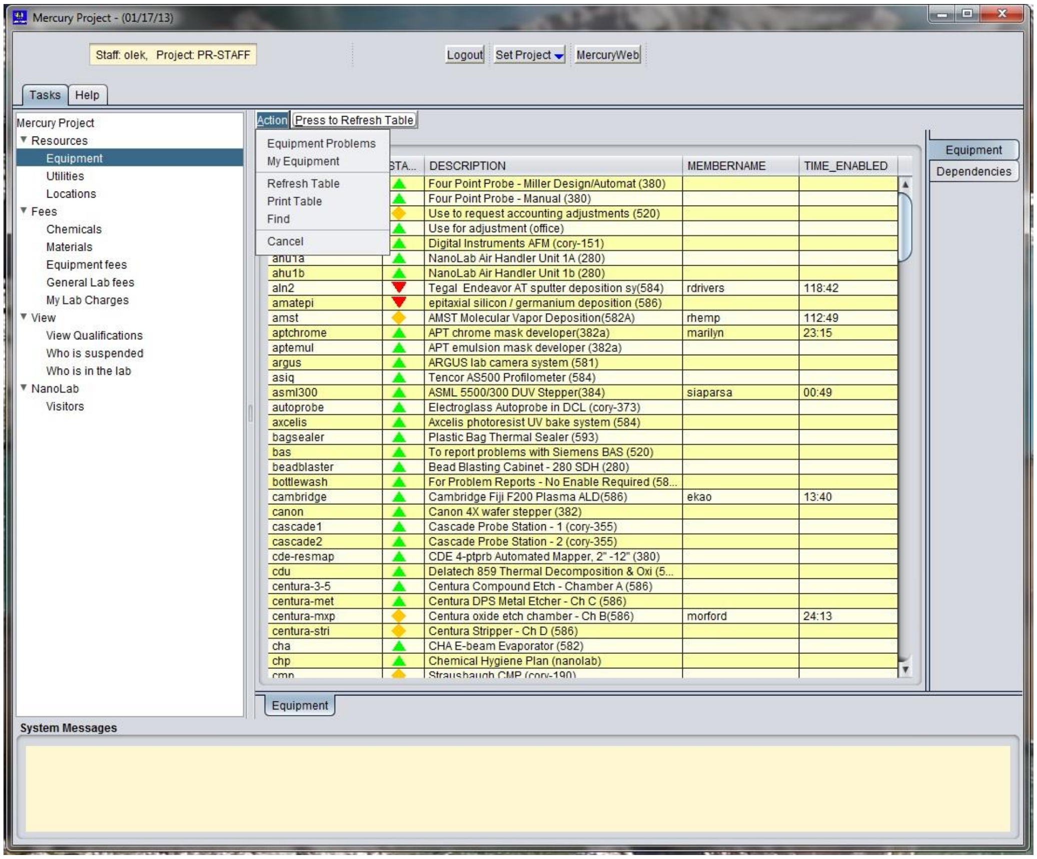
Fig. 31. Simplified Client Server Class Diagram

Once a client starts a session a Manager class keeps track of the session time and intervals between activities. If the client is inactive for a given period of time the Manager will logout the client session and send mail to the member indicating the that he was logged out automatically. The server also insures one session per user (gracefully logging out a member from an old session if a new session is started).

Remote objects are handled with a Request object wrapper class. This class simply holds the name of the object and parameters. It is packed at the client and unpacked at the server or vise-versus.

Mercury Client (GUI)

Fig. 32 shows the GUI based Mercury Client used by lab members and staff. The GUI is modeled based on current look-and-feels of pc-based applications; i.e. drop-down menus, multiple-panes, and tabs. Actions by the user proceed in a left to right fashion. On the left hand tree, a task is chosen that will bring up a populated table. An Action menu item is available to perform actions on the entire table such as printing or searching. If a row on a table is selected with a mouse click, numerous row based actions are available from enabling equipment to making a comment.



Mercury Project - (01/17/13)

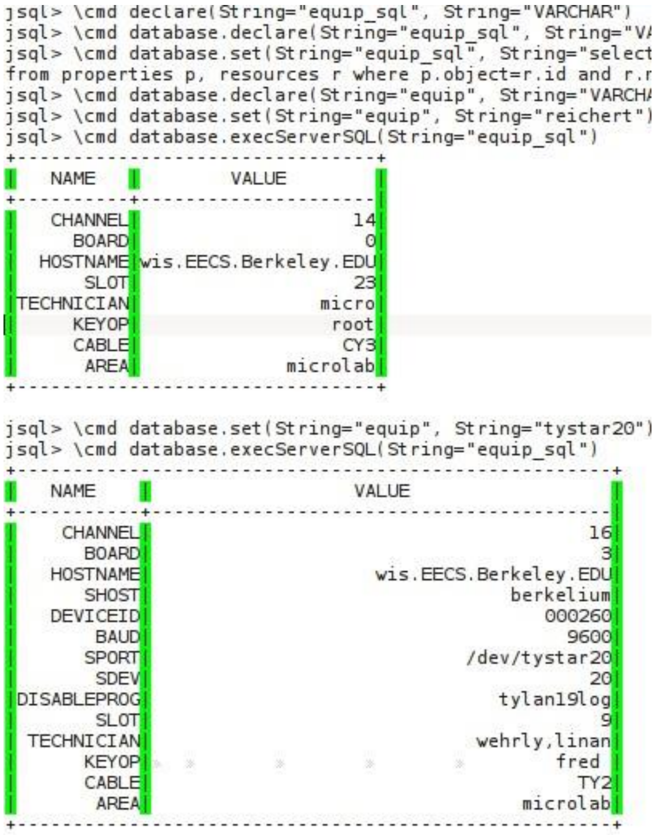
Staff: olek, Project: PR-STAFF

	STA...	DESCRIPTION	MEMBERNAME	TIME_ENABLED
	▲	Four Point Probe - Miller Design/Automat (380)		
	▲	Four Point Probe - Manual (380)		
	◆	Use to request accounting adjustments (520)		
	▲	Use for adjustment (office)		
	▲	Digital Instruments AFM (cory-151)		
ahu1a	▲	NanoLab Air Handler Unit 1A (280)		
ahu1b	▲	NanoLab Air Handler Unit 1b (280)		
aln2	▼	Tegal Endeavor AT sputter deposition sy(584)	rdrivers	118:42
amatepi	▼	epitaxial silicon / germanium deposition (586)		
amst	◆	AMST Molecular Vapor Deposition(582A)	rhemp	112:49
aptchrome	▲	APT chrome mask developer(382a)	marilyn	23:15
aptemul	▲	APT emulsion mask developer (382a)		
argus	▲	ARGUS lab camera system (581)		
asiq	▲	Tencor AS500 Profilometer (584)		
asml300	▲	ASML 5500/300 DUV Stepper(384)	siaparsa	00:49
autoprobe	▲	Electroglass Autoprobe in DCL (cory-373)		
axcelis	▲	Axcelis photoresist UV bake system (584)		
bagsealer	▲	Plastic Bag Thermal Sealer (593)		
bas	▲	To report problems with Siemens BAS (520)		
beadblaster	▲	Bead Blasting Cabinet - 280 SDH (280)		
bottlewash	▲	For Problem Reports - No Enable Required (58...		
cambridge	▲	Cambridge Fiji F200 Plasma ALD(586)	ekao	13:40
canon	▲	Canon 4X wafer stepper (382)		
cascade1	▲	Cascade Probe Station - 1 (cory-355)		
cascade2	▲	Cascade Probe Station - 2 (cory-355)		
cde-resmap	▲	CDE 4-ptprb Automated Mapper, 2" -12" (380)		
cdu	▲	Delatech 859 Thermal Decomposition & Oxi (5...		
centura-3-5	▲	Centura Compound Etch - Chamber A (586)		
centura-met	▲	Centura DPS Metal Etcher - Ch C (586)		
centura-mxp	◆	Centura oxide etch chamber - Ch B(586)	morford	24:13
centura-stri	◆	Centura Stripper - Ch D (586)		
cha	▲	CHA E-beam Evaporator (582)		
chp	▲	Chemical Hygiene Plan (nanolab)		
cmp	◆	Straushaugh CMP (cory-190)		

Fig. 32. Mercury Client.

Mercury Console Client

The jSQL is a command line terminal access console to the Mercury server and its back-end components. On a basic level SQL queries are possible, but more sophisticated use allows remote object invocation and flexible, expandable, automated testing of the server, database, and equipment control. Fig. 33 show a jSQL session executing a procedure similar to a Wand (the lab control program used in the Microlab) disable program. Commands to the jSQL program can be produced automatically using external scripts to automate system testing or create an input for another program; ie, the Unix way.



```
jsql> \cmd declare(String="equip_sql", String="VARCHAR")
jsql> \cmd database.declare(String="equip_sql", String="V
jsql> \cmd database.set(String="equip_sql", String="select
from properties p, resources r where p.object=r.id and r.r
jsql> \cmd database.declare(String="equip", String="VARCH
jsql> \cmd database.set(String="equip", String="reichert")
jsql> \cmd database.execServerSQL(String="equip_sql")
+--------------------------------+
|   NAME   |        VALUE        |
+----------+---------------------|
|   CHANNEL|                   14|
|     BOARD|                    0|
|  HOSTNAME|wis.EECS.Berkeley.EDU|
|      SLOT|                   23|
|TECHNICIAN|                micro|
|     KEYOP|                 root|
|     CABLE|                  CY3|
|      AREA|             microlab|
+--------------------------------+

jsql> \cmd database.set(String="equip", String="tystar20")
jsql> \cmd database.execServerSQL(String="equip_sql")
+----------------------------------------------------+
|   NAME    |                 VALUE                  |
+-----------+----------------------------------------|
|    CHANNEL|                                      16|
|      BOARD|                                       3|
|   HOSTNAME|                   wis.EECS.Berkeley.EDU|
|      SHOST|                               berkelium|
|   DEVICEID|                                  000260|
|       BAUD|                                    9600|
|      SPORT|                          /dev/tystar20|
|       SDEV|                                      20|
|DISABLEPROG|                              tylan19log|
|       SLOT|                                       9|
| TECHNICIAN|                            wehrly,linan|
|      KEYOP|                                   fred |
|      CABLE|                                     TY2|
|       AREA|                                microlab|
+----------------------------------------------------+
```

Fig. 33. jSQL Console example during development phase of MercuryServer (query of properties table).

Mercury Utility Applications

Aside for the jSQL client and Mercury Clent, utility based clients can be created by using Mercury Classes as a framework to develop applications. The code snipped in Fig. 34. shows one such application to send reservation reminders to lab members. Other applications in a similar vein include calendar reminders with maintenance postings and mail alias creation based on equipment engineers, and qualified members.

```
…..

public class MKReservedMail
    extends AbstractApplication {

    private Vector memvec = new Vector();
    public MKReservedMail() {
    }

    /**
     * populateMemberVector. Insert results from the reserve query into
     * the memvec vector. The vector is a collection of MemRes object.
     */
    public void populateMemberVector() {
        // query the database and populate vector.
        Integer rid;
        String today = "";
        int rows = 0;
        MemRes mr;
        DataSource query1;
        IDatabase database = ADatabase.getDB();
        java.util.Date dt;
        DateFormat plain = DateFormat.getInstance();
        // manipulate date string
        query1 = database.execute(sql_today.eval());
….
```

Fig. 34. Code snippet of Mercury utility program that makes use of Mercury Classes.

Mercury Server Communications to RUMS

When a member disables equipment that uses specialty gases, a property in the database tells the MercuryServer to query the Resource Monitoring System (RUMS) for data collected between enable and disable times and then calculate the volume of gas used, query the Resource table for the cost/volume, and create an entry in the activity table for the charge. The algorithm for gas volume calculation uses step interpolation (points are not equally spaced), and trapezoidal integration. An instance diagram in Fig. 35. steps through the gas acquisition and charge process in steps (vertical) and object (horizontal) interaction.

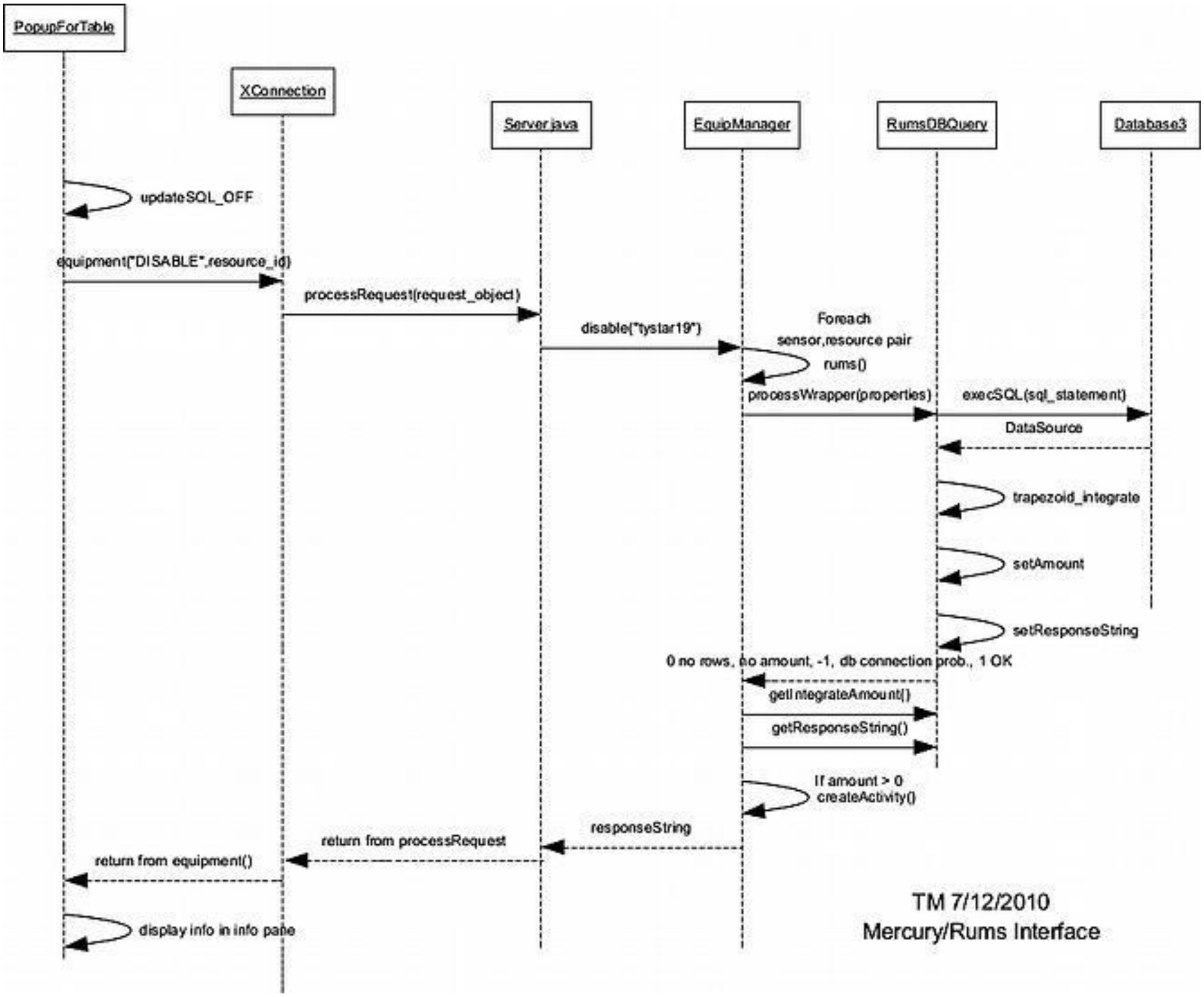


Fig. 35. Instance Diagram: a member disables a furnace.

Mercury Web

Mercury Web is a web application that provides lab members and staff access to the Mercury system through any web browser. Mercury Web is written in Java. It uses IceFaces JSF framework for the presentation layer, SQL queries and stored procedures to access and update data in the Ingres RDBMS, and the BIRT reporting engine which allows creating various reports in PDF, Word, Excel, and PowerPoint formats. Mercury Web runs under Apache Tomcat and Apache httpd server; the latter serves as an additional level of security. Mercury Web includes the following major modules: Accounting (Fig. 36), Inventory, Member Management, Online Tests (Fig. 37), Facilities (Fig. 38), Reservations, Calendar, and Tasks.

The Accounting module is used for day to day tasks, such as crediting, debiting, and updating transactions, defining and editing fund sources. It also provides various monthly and yearly financial statements and reports. Member and staff account setup and administration is also performed using accounting module. (Fig. 36.)

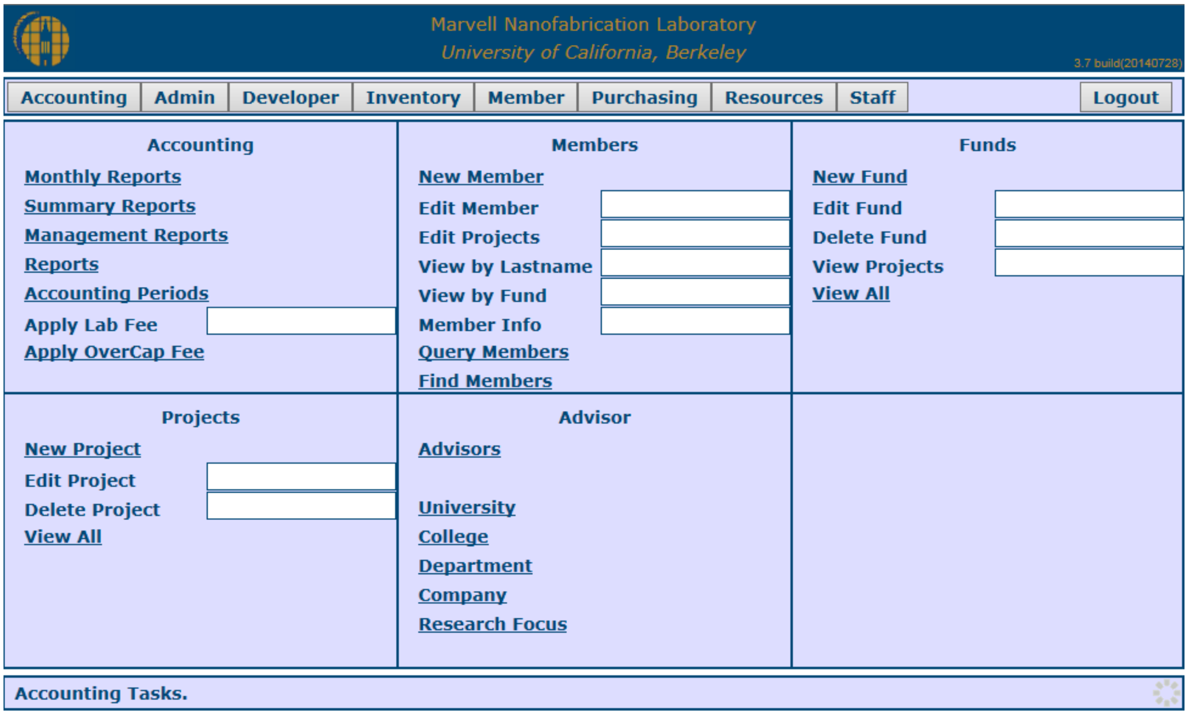
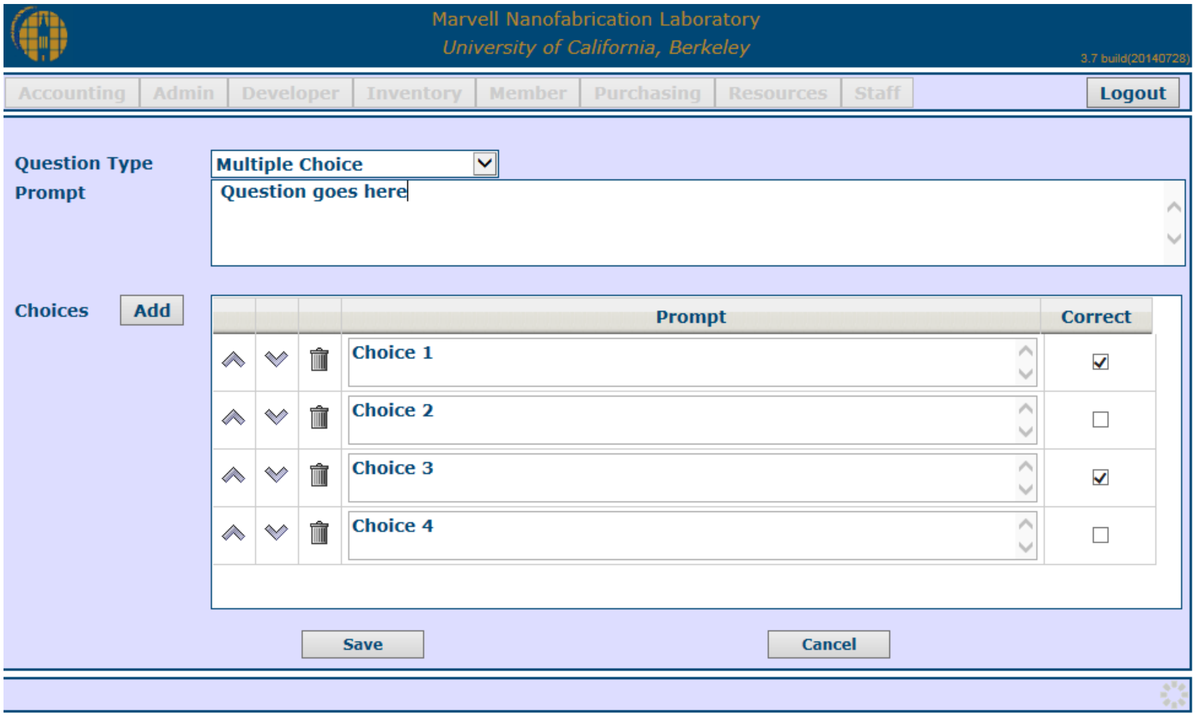


Fig. 36. Mercury Web main Accounting page.

Online Tests module allows creating, taking, and grading tests online, completely replacing paper based tests. When creating test one can select from various question types such as multiple choice, single choice, true/false, essay, mandatory (failing to answer this question automatically fails entire test) questions. If the test does not have any essay questions it is graded automatically and members know their result immediately. Otherwise notification email is send to the designated grader. Once the test is graded member is notified about results by email. (Fig. 37.)



Question Type: Multiple Choice

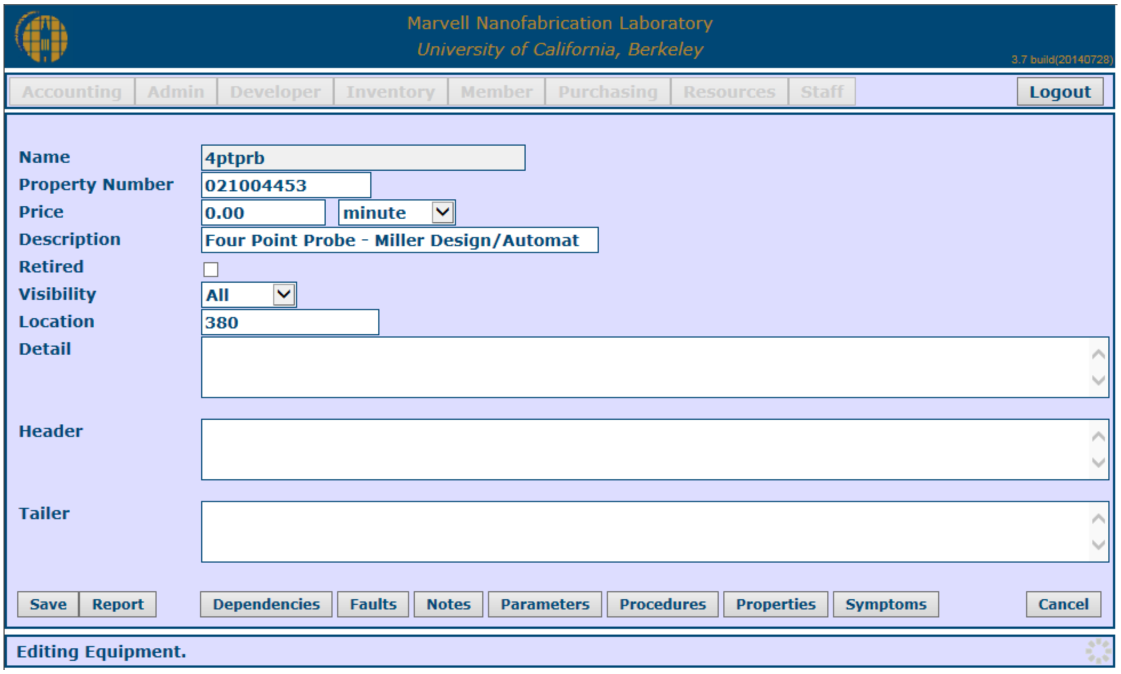
Prompt: Question goes here

Choices

Prompt	Correct
Choice 1	☑
Choice 2	☐
Choice 3	☑
Choice 4	☐

Fig. 37. Configuration page of Online Test.

The Facilities module is used to define resources (equipment, utilities, and locations) and create associations between them. (Fig. 38.)



Name	4ptprb
Property Number	021004453
Price	0.00 minute
Description	Four Point Probe - Miller Design/Automat
Retired	☐
Visibility	All
Location	380
Detail	
Header	
Tailer	

Editing Equipment.

Fig. 38. Creating new equipment.

The Staff page provides quick links to most common task performed by staff. (Fig. 39.)

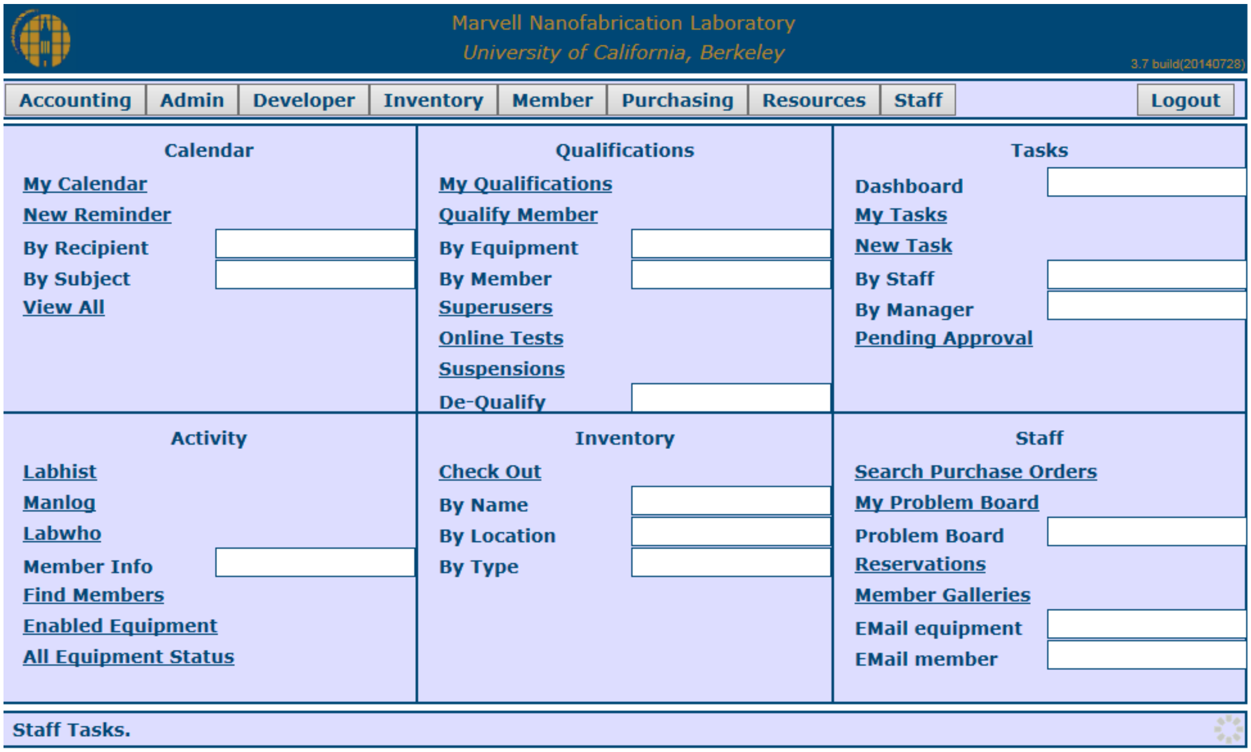


Fig. 39. Staff page of Mercury Web.

Developer Tools

CVS, http://www.gnu.org/software/cvs
Ingres, http://community.actian.com/wiki/Ingres_DBMS_Home
Ingres Documentation http://docs.actian.com/
nanoxml,http://sourceforge.net/projects/nanoxml
beanshell,http://www.beanshell.org
netbeans, http://www.netbeans.org
BIRT, http://www.eclipse.org/birt
Java, http://www.oracle.com/us/technologies/java/overview/index.html
Apache, http://www.apache.org
Tomcat, http://projects.apache.org/projects/tomcat.html
Sendmail, http://www.sendmail.org
IceFaces, http://www.icesoft.org/java/home.jsf
Make, http://www.gnu.org/software/make/manual/make.html

System Requirements

These requirements do not address individual user disk storage, office-type applications, and other run-of-the mill computer requirements.

Hardware-Server: Servers can be distributed and linked through network protocols. A Linux based set of servers is probably the most economical. The number and power of each server is dependent on institutional needs. Servers can be setup in multiple roles or as fallback in case of failure. Specifics for the NanoLab are shown in figures 40 and 41. Servers need to run the following applications:

Relational Database Management System: Ingres
Web Server: Apache with Secure Socket Layers
Servlet Container and Web Server: Apache Tomcat
Java Runtime Environment and Development Kit to run Mercury Server
Mail Server: SendMail (Linux/Solaris)

Keep in mind that each system will require disk and power supply redundancy and off-site backup.

Hardware-Client: Windows Terminal Server
(authenticated sessions need to be active as members move from terminal to terminal).
Software-Java Client: Java Run-time Edition

Building, Distribution, and Revision Control

Mercury is a well-engineered system. The team that created it intensely debated, tested features, technologies, look and feel, and other aspects of the system. Overall, the design is based on the successful and long-lived BCIMS system (operations of the Microlab) and industry best practices. As any engineered product, successful maintenance after the hectic release requires attentive and experienced staff to apply fixes and refinements, and knowledgably advocate for the system's health. Building the system should only be taken on by journeymen level programmers. It goes without saying that the software industry has very short product cycles. New installations may require fixes just to manage components that are no longer supported, have security issues, or are soon to be deprecated.

Code developed for the project is kept under revision control. These files are organized in a directory hierarchy based on their functionality. Since Mercury is composed of a variety of components, some third party tools used in Mercury are not maintained by the developers. The tools are maintained and distributed on their project web sites.

Files are kept under source control using CVS (Concurrent Version System). Developers keep a local copy of these files for editing and testing. They will then check the files back into the repository. Mercury can be developed and run on Windows, Solaris, and Linux based platforms. CVS nicely plugs in to the Netbeans and Eclipse (IDE) Integrated Development Environments.

Mercury System CVS Repositories

```
~mercury/
          | ----- Mercury                  # Mercury Client/Server/Util applications
          | ----- Mercury Web              # Web Based applications build with IceFaces 1.6
          | ----- MercuryIce               # Web Based applications build with IceFaces 3.1
          | ----- Mercury WebCommon             # Common library for Mercury Web and
MercuryIce
          | ----- Mercury WebReports       # BIRT reports for Mercury Web
          | ----- MercuryIceReports        # BIRT reports for MercuryIce
          | ----- cvs_emitter              # CVS emitter for BIRT
          | ----- ErrorValve               # Mercury Web specific error valve for Tomcat
          | ----- IceFaces                 # Custom version of IceFaces 1.6.2
```

Mercury Repository

```
Mercury
          |----- database         # Ingres schema and utilities
          |----- src              # Mercury Client/Server/Util source
          |----- lib              # third party class and jar files
```

Mercury Production Server Directory Structure

```
server
          |----- config           # properties files and xml (server only)
          |----- lib              # class and jar files
          |----- bin              # startup and utility scrips
          |----- store            #
```

Mercury Production Client Directory Structure

```
client
          |----- config           # properties files and xml (server only)
          |----- lib              # class and jar files
          |----- bin              # startup and utility scrips
          |----- resources        # icons
```

The build for distribution/production takes place on Unix Systems using the "ant. The following targets are available:

```
build.xml
        |----- compile          # compiles all source files
        |----- jar              # makes all required (server, client, and util) jar files.
        |----- server_dist      # creates server distribution
        |----- client_dist      # creates client distribution
        |----- full_dist        # creates both client and server distribution
        |----- clean            # cleans build directories (class and jar)
        |----- dist_clean       # in addition to executing clean target cleans dist directories.
```

Target directory where distribution (client or server) is installed is controlled by the following properties defined in build.xml

```
        <property name="server_prefix" value="/usr/local/NanoLab/home/mercury" />
        <property name="client_prefix" value="r:\\Mercury" />
```

Mercury Web Repository:

Mercury Web is a Netbeans project, so it follows Netbeans 5.5 Visual Web Project directory structure.

MercuryIce Repository:

MercuryIce is an Eclipse project, so it follows Netbeans 5.5 Dynamic Web Project directory structure.

Mercury WebCommon Repository:

MercuryCommon is a Netbeans project, so it follows Netbeans 5.5 Java Class Library directory structure.

The Mercury Client on Windows

There is a batch file that can be used to start the mercury client. It is client.bat. This script is automatically generated during the build process.

Computers used by NanoLab staff, that use Microsoft Windows operating systems are part of the departments “Active Directory” structure that manages identities and resources on the network. As such the computers in the NanoLab are part of an organization unit under the EECS domain. In the EECS domain, a group policy is set so that all client computers mount the R: drive automatically that is on one of the NanoLab Window's Servers. R: to \\NanoLab2\Mercury. The Windows Mercury distribution is copied to this location.

In the NanoLab, members connect to a Windows terminal server, CAPE, with their unique account and start up their Mercury Clent sessions.

The Windows Mercury distribution is copied to C:\Mercury\ on CAPE2.

NOTE: any host connecting to Mercury requires its IP address registered in the Mercury database "hosts" table.

The Client or Server on Solaris or Linux

Starting the server on Linux or Solaris is done automatically during the boot process by /etc/rc3.d/S90mercury. The same script is used to shutdown server on system reboot or shutdown. Alternatively the script can be run manually with start or stop parameter to respectively bring up or down the server.

mercury_util: a wrapper script to start various Mercury utility programs.

Unix startup scripts read CLASSPATH and other distribution information from a file called mercury_env. mercury_env is created from the build process.

Some programs require a user name password to initiate utility program or to use as an unauthenticated user. These data are kept in a file with the password as an encrypted version of the system password. The password is later decrypted on MercuryServer.

Mercury requires the use of "Sendmail" Mail Transport Agent. The server has an object available to the client to send messages and will send a message if a session expires through a time-out.

Executable utilities (compiled from java) are run daily to generate aliases maintanence messages, and reservation reminders. These programs require Sendmail as well.
(MKCalendar, MKAlias, MKReservedMail)

Properties and Configuration Files used on Mercury

mercury_root/resources/client.conf # Server addresses and ports.
mercury_root/config/server.conf # Final Variables for the server: timeouts, paths, ports, and classes.

Building and Using the Ingres Relational Database System.

Downloading, Installing, Configuring,

The Ingres database can be downloaded from www.actian.com

Important install options.

1) Use ingres date
2) Do not use strict ANSI/ISO compliance.
3) JDBC connection is through the DataAccessServer

Since there are various software components in Mercury, appropriate versions of the jdbc driver may be required for compatibility.

Creating and Populating the Mercury Database

Building database tables and procedures are dependent on ordering. Bigbang may not work unless sources are properly ordered to consider dependencies. Bigbang destroys the database, creates the database, and then adds tables, procedures, and data.

1) cd database/schema; gmake bigbang

2) To create groups

```
./mercury_util groups create
```

3) To create appropriate grants on tables and stored procedures

```
cd database/schema/access; gmake access
```

Note some procedures and tables will have data preloaded. This is lab specific.

Hardware

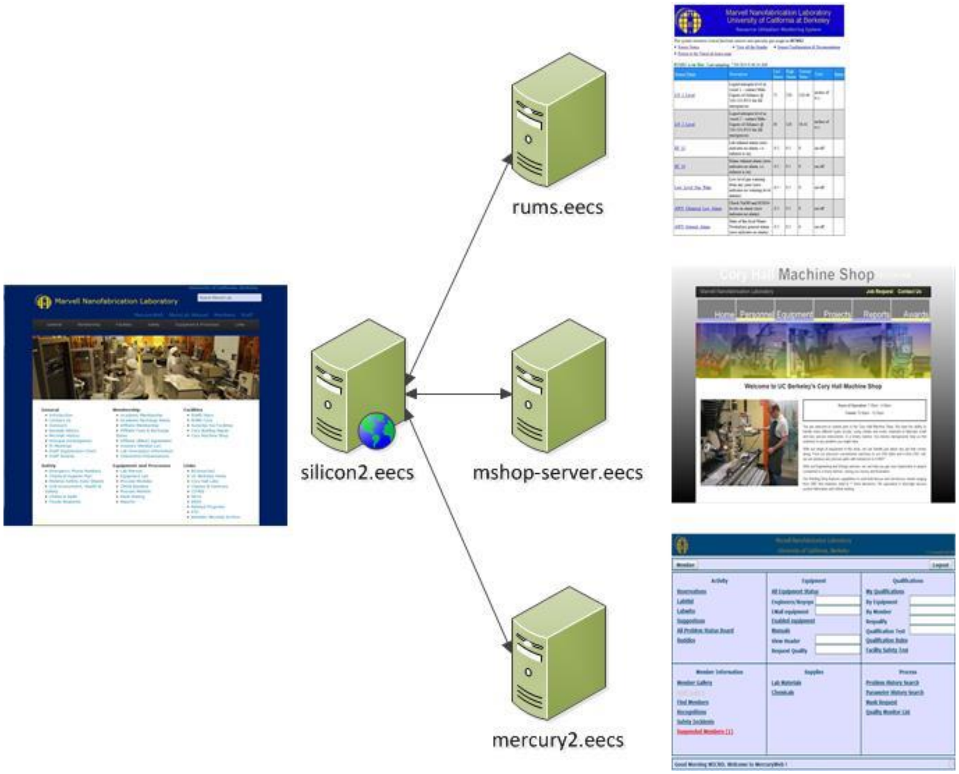

Marvell Nanofabrication Laboratory
University of California at Berkeley
rums.eecs
Machine Shop
Welcome to UC Berkeley's Cory Hall Machine Shop
silicon2.eecs
mshop-server.eecs
mercury2.eecs

Fig. 40. NanoLab web servers.

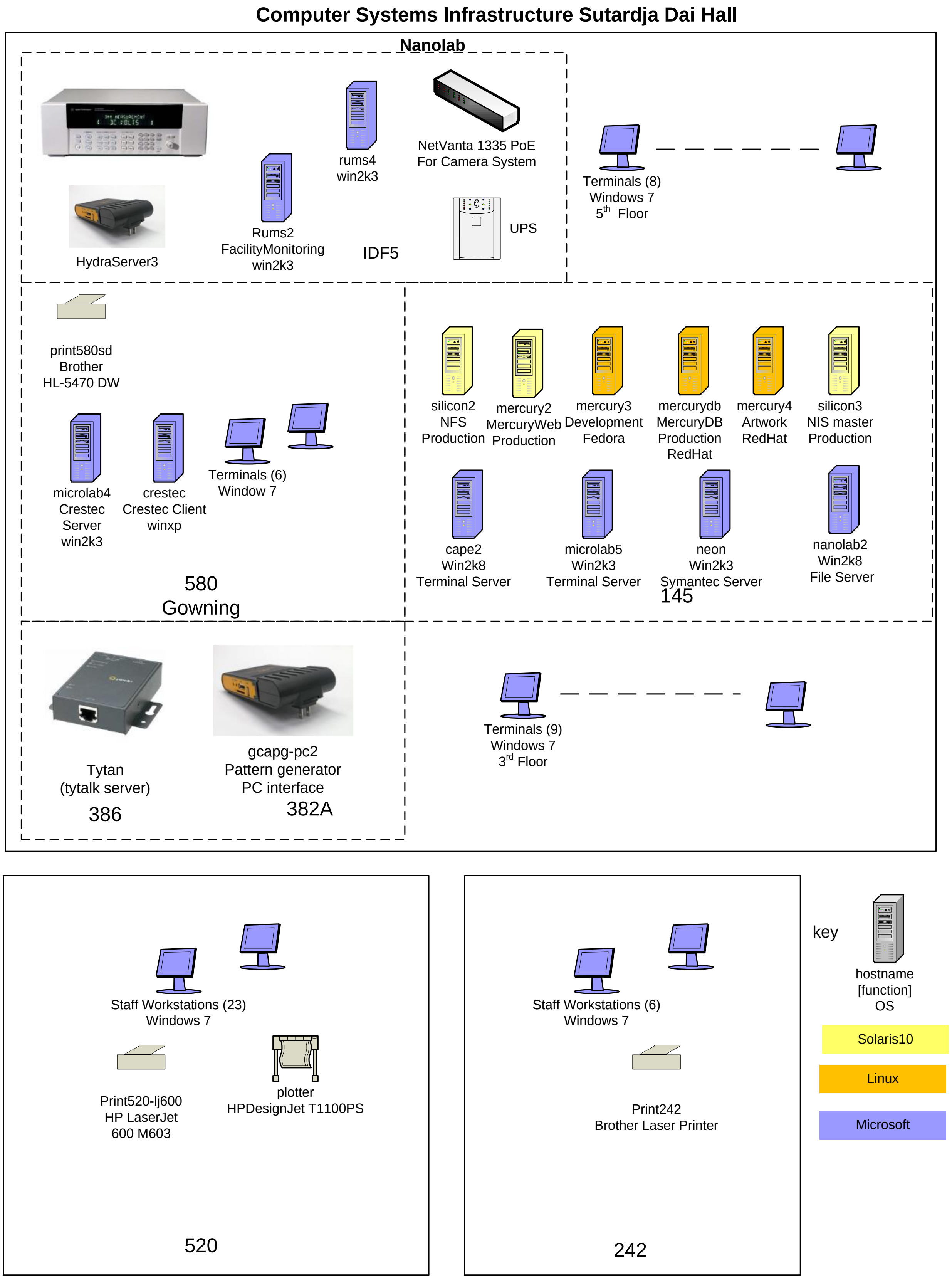


Fig. 41. NanoLab computer infrastructure in 2014.

Independent Modules Connected to Mercury

Equipment Interlock System – Hydra

The equipment control system hardware is based on the Agilent 349080A Multifunction Switch/Measure unit equipped with multiplexing high-density magnetic latching relays. The system is configured to send a pulse to an addressed channel which is connected to a Hydra interlock box. The 349080A has a serial, GPIB, and network interfaces allowing for very flexible operation. The software interface between Mercury and Hydra is part of the Mercury system complex. (Fig. 42.)

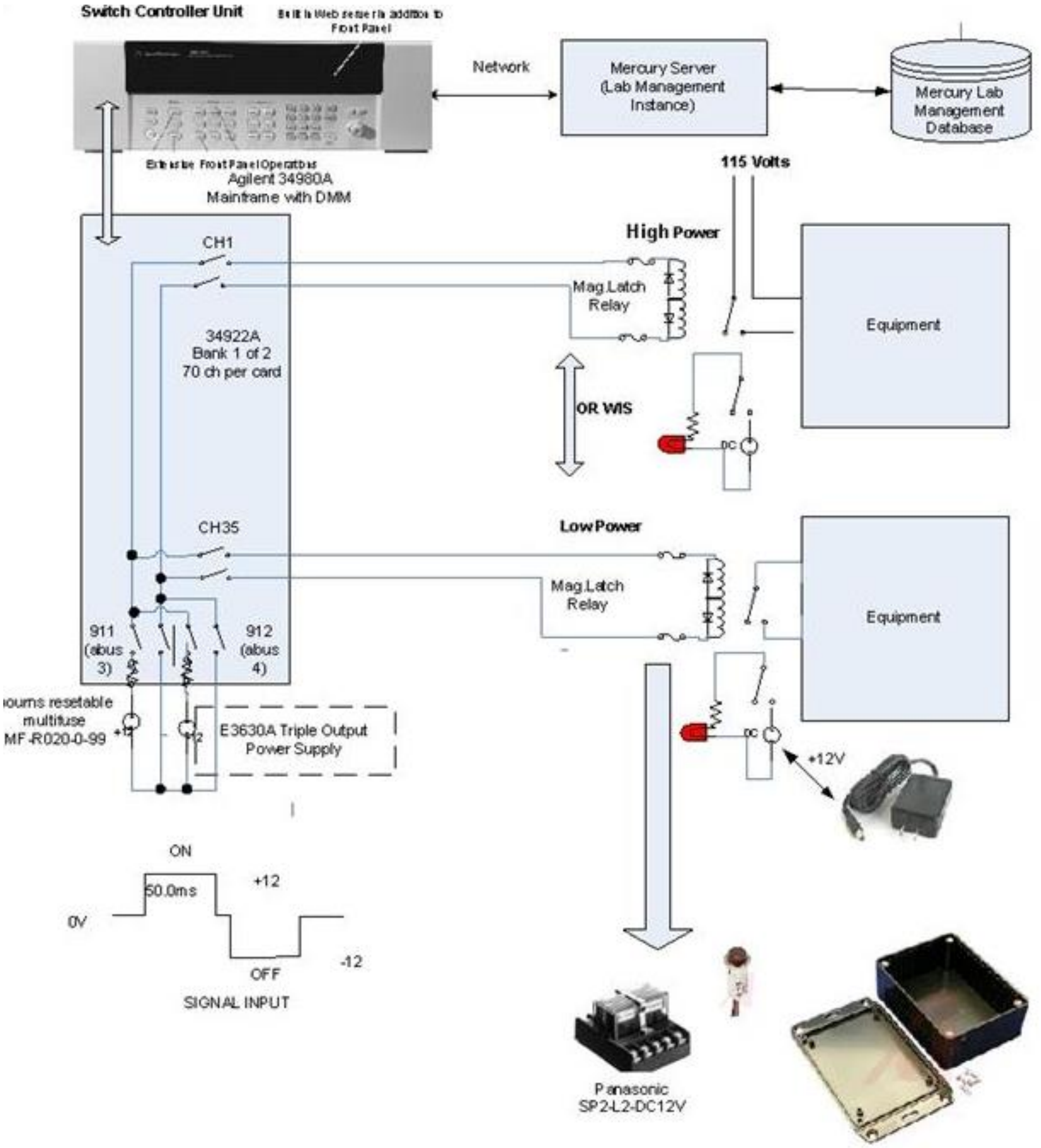


Fig. 42. Hydra equipment control system.

Gases Database

Semiconductor processing in the lab requires 46 different types of gases; 95-100 cylinders of specialty gases are in use at any one time, and 60 cylinders are spare stock. Because of its complexity the **Gas Database** is separate from the general parts inventory database, accessible by an interactive Objects-by-Forms system by staff. (Fig. 43) The program utilizes Microsoft Access on an SQL (Structured Query Language) server, to present a web-based interface for the viewer and interactive access to the inventory manager.

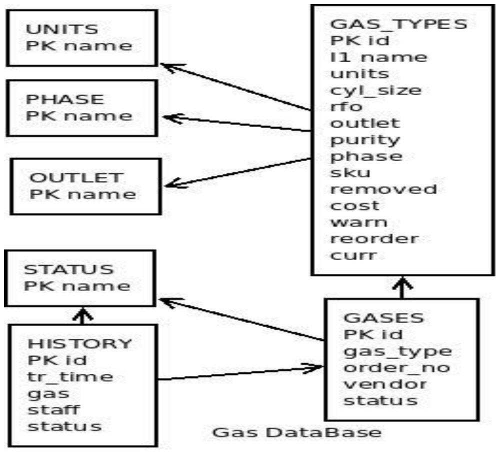


Fig. 43. Gas Database schematic.

Utilities Monitoring System and Database

An important aspect of managing a semiconductor laboratory is tight control of the laboratory environment and utilities needed to operate processing equipment. RUMS, a Resource Utilization Monitoring System, was developed as a separate, standalone system, which is closely connected to Mercury. RUMS provides Mercury with vital environmental data, which Mercury processes as part of the equipment control program. Simply, Mercury will not allow use of a tool unless environmental specifications are met. RUMS was the subject of an earlier report by T. Duncan, T.K. Chen, D. Pestal and T. Merport [3] and will not be detailed here; an example of the graphical user interface (GUI) is shown in Fig. 13. The schematic outline of the components is presented in Fig. 44.

RUMS employs a National Instruments data acquisition card in a dedicated PC with Windows 2000 platform, connected by Ethernet to the lab's main server. Data is transmitted to the RUMS computer from a variety of locations by either current sensors or contact closure sensors, through direct wiring to a connector box and the PC. Data management, displays, and alarm emails were provided by the Rums Server application software, utilizing National Instruments' LabVIEW.

A huge portion of the development of RUMS was compiling an accurate database of an equipment and utilities matrix. Utilities dependencies were determined for each piece of equipment and documented in the RUMS database. Data includes entries for all tools, each depending on some of the 25 utilities needed to maintain the lab. When a utility failure occurs, the program sends an immediate alarm to assigned staff and users of the machines affected.

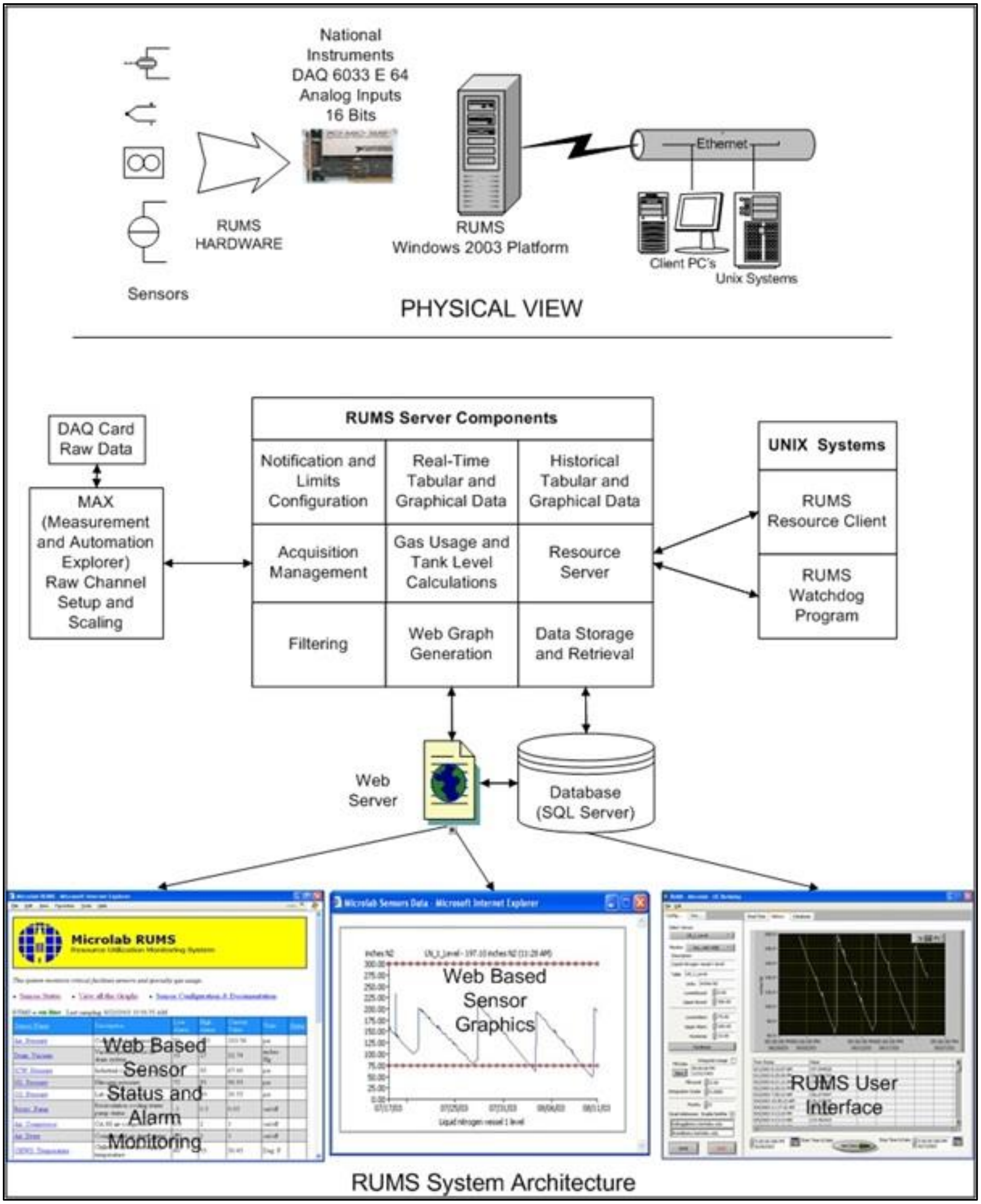


Fig. 44. Resource Utilization Monitoring System, RUMS [3].

VI. Summary

Mercury is a well-engineered system. The team that created it intensely debated, tested features, technologies, look and feel, and other aspects of the system. Overall, the design is based on the successful and long-lived BCIMS system (used in the Microlab) and industry best practices. The dual system of Mercury Client for using equipment and processes inside the lab and of Mercury Web for non-critical activities simplifies and streamlines computer operations and provides additional security. Information is stored in a common database and retrieved by each module as needed.

The Mercury database was designed with both relational database and object oriented design patterns. The object patterns provide inheritance such as equipment "is a" resource and polymorphism where table rows are interchangeable objects that can be passed to procedures, grouped, or queried as needs arise. Relational systems provide a high degree of organization, data integrity, standards, and maturity. The goal is a fast, reliable, and flexible system.

The Accounting module is used for day to day tasks as well as to create end of month financial statements and reports. The Inventory module helps to maintain the inventory of supplies and parts used in the lab. Member Management provides member and staff account setup and administration. Online Tests allow creating, taking, and grading tests online, completely replacing paper based tests. Facilities are used to define resources (equipment, utilities, and locations) and create associations between them. The Reservation modules allow lab members to reserve frequently used equipment.

The Mercury system has built in flexibility to enable simultaneous operations of multiple facilities, each with its own equipment, facility and charge rules. It is capable to produce separate or merged financial reports. Maintenance of the Mercury system, operations in the Berkeley NanoLab with over 500 individual yearly accounts and a staff of 26 employees, requires two full time programmer/analysts. After four years of real time operation the system is running at 99.9% uptime.

VII. References

[1] K. Voros, "History of the UC Berkeley Microlab," Technical Report No. UCB/EECS-2013-158, University of California, Berkeley, September 2013.

[2] L. J. Massa-Lochridge, "BCIMS: The Berkeley Computer Integrated Manufacturing System," ERL Memorandum No. UCB/ERL M95/46, University of California, Berkeley, June 1995.

[3] T. Duncan, T.K. Chen, D. Pestal and T. Merport, "RUMS - Resource Utilization Monitoring System," ERL Mamorandum No. ECB/ERL 03/43, University of California, Berkeley, November 2003.

[4] D. C. Mudie, "FAULTS: An Equipment Maintenance and Repair Tracking System Using a Relational Database," ERL Memorandunm No. UCB/ERL M91/44, University of California, Berkeley, May 1991.

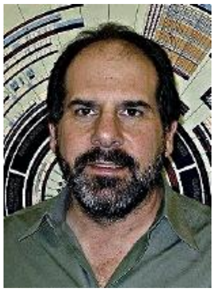
Acknowledgements

The authors thank Prof. David Hodges for initiating and supporting the project of computer control in the Microlab, from the time that the lab was built in 1982. The original software, developed by students, was further enhanced by Microlab staff. As the technology developed, upgrades were frequently and scrupulously made by the lab's computer staff who applied new software tools to keep the system up-to-date. The original system, BCIMS, ran for 28 years and was the basis for the design of the Mercury system. Prof. Hodges' interest and mentorship is hereby gratefully acknowledged.

Several programmers designed and coded Mercury or helped lay its foundation, particularly David Mudie, Ferenc Varju, Tim Duncan, and Eniko Seen. During the design phase inputs from Bill Murray and John Shott at Stanford were valuable. Some of their ideas such as multiple projects and utilities disabling dependent equipment made it into Mercury.

Additionally, input from Microlab staff members provided clarity that was crucial to Mercury's adaptation. Special thanks are due to Rosemary Spivey and Susan Kellogg-Smith. They provided detailed information, testing and feedback during the development of the Accounting module. Bob Hamilton provided input from the facility/maintenance point of view and filtered suggestions from his staff. Bill Flounders supplied useful suggestions as the project went along and quickly became an expert user once it was released. He then added modifications, fixes as needed by the NanoLab. We thank you all.

Biographies



Todd Merport has worked in electronics and software beginning in 1982 when he repaired circuits at G. E. Intersil systems. He later worked as an Engineering Technician for Dalmo Victor (a defense contractor) bread-boarding and testing high-voltage power supplies and radar simulators. He has also worked for Agilent Technologies and U.C. Berkeley. At U.C. Berkeley Todd worked for the Civil and Environmental Engineering Department as a Senior Development Engineer and at the Microlab/NanoLab managing computer systems. Currently, Todd develops code for Android based mobile devices. Todd was awarded the Chancellor's Special Achievement Award in 1991.

Olek Proskurowski started his studies at the Warsaw University of Technology. He graduated from the University of Southern California in 1993 with a B.S. degree in Computer Science. He worked as a Software Engineer for IA Corporation and also at CSC Corporation, Oakland, CA, as a Senior Software Engineer. He joined the UC Berkeley Microlab in 2006; he is now the Computer Systems Manager for the Marvell Nanofabrication Laboratory.

Katalin Voros graduated from the Drexel Institute of Technology (Philadelphia) in 1966 with a B.S. degree in Physics. She worked as a process engineer for Philco-Ford Microelectronics (bipolar ICs), Solid State Scientific, Inc. (RF transistors, CMOS circuits), and Microwave Semiconductor Corporation (high frequency power transistors). She joined RCA's David Sarnoff Research Center in Princeton, New Jersey in 1980 as an associate member of technical staff, in high density bulk CMOS (SRAM) development. In 1984 Ms. Voros received her MS degree in Engineering Science in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, where she was retained in the Microfabrication Laboratory as a process development engineer. A Principal Development Engineer, Ms. Voros was Operations Manager of the UC Berkeley Microlab, from 1986-2010. She also participated in an inter-departmental research group studying competitive manufacturing in the semiconductor industry. She retired as R&D Engineering Manager in June 2013.

Ms. Voros is a member of the Electrochemical Society, and a Senior Life Member of IEEE. She is a recipient of the University of California, Berkeley, Administrative & Professional Staff Distinguished Achievement Award (1991), the Berkeley Staff Assembly's Excellence in Management Award (1995 and 2001) and the Chancellor's Outstanding Staff Award (2011). She received the Gold Cross of Merit of the Republic of Hungary in 2009. After 30 years with the University of California, Berkeley, upon retirement Katalin Voros had been conferred the title R&D Engineering Manager Emerita.

Appendix

Mercury XML Documents

Description

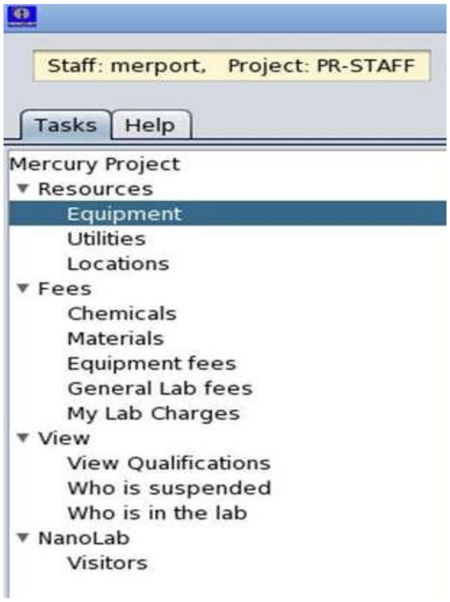
The Mercury software system uses XML documents to dynamically generate Java code that once executed will generate GUI components, fields and database queries. This paradigm allows user interface elements to be updated without recompiling the source code (a great help for developers). XML data can be fetched from flat files or stored in a database.

The most common use of XML documents in Mercury is a Client request to the Server to supply Java code as a script. The Client will execute the script. Here is an an example of how XML documents are used when the Client logs into Mercury Client and selects an item from the left hand tree.

When the member is authenticated the document MAIN_WINDOW.xml is fetched from the Mercury Server. The user interface is generated.
A code fragment from MAIN_WINDOW:

```
<?xml version="1.0"?>
<script>
...
private String[][] treeObjects = {
      {"member", "Tasks"},
      {"member_admin", "Administration"}
  };
private Object[][] member = {
      { "Resources" } ,{
      new URLInfo( "Equipment", "TAB_EQUIPMENT"),
      new URLInfo( "Utilities", "TAB_UTILITIES"),
      new URLInfo( "Locations", "TAB_LOCATIONS")},
      { "Fees"}, {
      new URLInfo( "Chemicals", "CHEMICALS"),
      new URLInfo( "Materials", "MATERIALS"),
      new URLInfo( "Equipment fees", "EQUIPMENTFEES"),
      new URLInfo( "General Lab fees", "FEES"),
      new URLInfo( "My Lab Charges", "MYLABCHARGES")},
      { "View"}, {
      new URLInfo( "View Qualifications", "QUALIFICATIONS"),
      new URLInfo( "Who is suspended", "SUSPENSIONS"),
      new URLInfo( "Who is in the lab", "LABWHO")},
      {"NanoLab"},
      { new URLInfo("Visitors", "VISITORS")}
  };
...
</script>
```

The Client Will populate its tree nodes and leafs with the objects described by the strings “Resources”, ”Fees”, ”View”, “NanoLab”.



Staff: merport, Project: PR-STAFF

From there the member can select an item from the tree. In the screen shot above “Equipment” is selected. This is handled by the Mercury Client in the MJTreeTable class. The Client passes the node name to the server and retrieves the associated XML document.

Code segment from MJTreeTable Client Class:

```
...
 private void myLeftMouseSingleClick(DefaultMutableTreeNode aNode) {
      nodeInfo = aNode.getUserObject();
...
      if (aNode.isLeaf()) {
         URLInfo urlLink = (URLInfo) nodeInfo;
         if (urlLink.xmlFileName != null) {
            // create table view
            try {
               (ClientApplication.getInterpreter()).set("componentTitle",
                     nodeInfo.toString());
            } catch (EvalError ee) {
               System.out.println(ee);
            }
            String getDocument = XConnection.getConnection().getXML(
                  urlLink.xmlFileName);
            if (getDocument != null) {
               ClientApplication.evaluate(getDocument, this);
….
```

Below is a sample of the XML document, TAB_EQUIPMENT, sent by the server. It has the information needed to populate the “Equipment” table (to be seen by non-staff only).

```
<?xml version="1.0"?>
<script>

<table id="equipmentTable_m" name = "Equipment" >
   <sql id="equipmentTable_mSQL">
      SELECT r.name,r.status ,r.descrip + ' (' + r2.name + ')' as DESCRIPTION,e.membername, NULL as
time_enabled, r.id
      FROM resources r, equipment e, resources r2
      WHERE e.id=r.id
      AND  e.location = r2.id
      AND r.retired = 'n' AND r.visibility = 'm'
      and e.membername is null
      UNION
      SELECT r.name, r.status, r.descrip + '(' + r2.name + ')' as DESCRIPTION, e.membername,
      INT4(INTERVAL ('minute',DATE ('now') - DATE (entry_time))),r.id
      FROM resources r, equipment e, resources r2, activity a
      WHERE r.id = e.id
       AND   r.retired = 'n'  AND   r.visibility != 's' AND   e.location = r2.id AND   e.id = a.resource
      AND a.status = 'o'
      ORDER by r.name
   </sql>
</table>
</script>
```

Once TAB_EQUIPMENT.xml is loaded and executed, the right table on the Client user interface is populated with equipment items, shown in Fig. 8.

There are two widely accepted open source software tools (along with Java) that Mercury utilizes to implement this task. The first is a lightweight XML parser, NanoXML; the second is a Java scripting language BeanShell (which is to be included in the formal Java specification shortly). To create the dynamically generated Java code, a plain-text XML document is parsed with the XML parser. The parser notifies the IXMBuilder interface (which is registered with the parser) of events while parsing the document such as starting or ending of an element or attribute. Once the document parsing is completed, the results from the builder — Java language source code — are interpreted by the BeanShell interpreter. The Mercury system implementation of the IXMLBuilder interface is a class ScriptBuilder.

Grammar

Described below is a Document Type Definition (DTD) or language grammar for Mercury’s XML documents.

```
<ELEMENT “root” (object+)>
<ELEMENT object >
<!ATTLIST object
id  CDATA #IMPLIED
class CDATA #REQUIRED
set-accessor CDATA #IMPLIED
            script CDATA # IMPLIED >
<!ELEMENT script CDATA  #REQUIRED>
object: one of
object
alias
alias:  one of
ScriptBuilder.CLASSES[]
set-accessor:
            action  text name default title label …
```

Note on typography:

Constant-width is used for literals.
Bold-italicized is used for identifiers.

The first line of the DTD is implied and not actually used in the XML document.

The ScriptBuilder.CLASSES[] array is specified in the configuration file gui.conf.

If ***alias*** is used as ***object***, the class attribute is not required.
set-accessor is the suffix of a set accessor used for the actual Java class to which the element refers. **set** is the implied prefix of the method name. The table below shows more detail.

Attribute *`set-accessor`*	**Java**
action = ”$`value`”	setAction(value);
text = “`value`”	setText(“value);
name = “`value`”	setName(“value”);
default = “true”	setDefault(true);
script = “doLogin”	setShell(**shell**); setScript(“doLogin()”);

If value has a $ prefix or is specified as true or false, quotes will be removed in the corresponding Java function. If the attribute is script, an additional line of Java, setShell(shell) is created before the setScript(“value”) method is called. The shell is a Bean Shell interpreter object instance. A null attribute value will result in an empty parameter in the resulting Java method.

Every element's attribute id *value* is used as a parameter for the parent elements add method (see Example 1). The last line of the XML generated script adds the first element parsed to the to a named object root (see Example 1). The BeanShell interpreter has a handle to this object. If the attribute id is not specified, an object reference will be generated automatically by the ScriptBuilder.

This last example shows how a simple XML document is parsed when used to generate a user interface component. In this case a Send Mail dialog:

XML	Java

<?xml version="1.0" ?>
<window title="Send Mail" resizable="true">
<form id="form1" clear="true">
 <object class="XLabel"
label="From"><macro name="from"
id="from" value="mercury@silcion"
writable="false" /></object>
<input label="To"><macro name="to"
mandatory="true" id="to" /></input>
<input label="Subject"><macro
name="subject" id="subject" /></input>
<edit label="Message"><macro
name="message" id="data" /></edit>
<cancel text="Cancel" />
<submit text="Send" default="true"
script="connection.sendMail(to.getValu
e(), subject.getValue(), data.getValue())"
/>
</form>
</window>
	import Java.text.*;
	import Client.gui.*;
	import Client.app.*;
	import common.remote.Request;
	import Client.connect.ADatabase;
	import common.data.DataSource;
	import common.data.SQL;
	
	Client.gui.XWindow window_1 = new Client.gui.XWindow();
	window_1.setTitle("Send Mail");
	window_1.setResizable(true);
	Client.gui.Form form1 = new Client.gui.Form();
	form1.setClear(true);
	XLabel object_2 = new XLabel();
	object_2.setLabel("From");
	common.data.ObjectMacro from = new
	common.data.ObjectMacro();
	from.setName("from");
	from.setValue("mercury@silcion");
	from.setWritable(false);
	object_2.add(from);
	Client.gui.XTextField input_3 = new Client.gui.XTextField();
	input_3.setLabel("To");
	common.data.ObjectMacro to = new
	common.data.ObjectMacro();
	to.setName("to");
	to.setMandatory(true);
	input_3.add(to);
	Client.gui.XTextField input_4 = new Client.gui.XTextField();
	input_4.setLabel("Subject");
	common.data.ObjectMacro subject = new
	common.data.ObjectMacro();
	subject.setName("subject");
	input_4.add(subject);
	Client.gui.XTextArea edit_5 = new Client.gui.XTextArea();
	edit_5.setLabel("Message");
	common.data.ObjectMacro data = new

	common.data.ObjectMacro();
	data.setName("message");
	edit_5.add(data);
	Client.gui.Form.CancelButton cancel_6 = new Client.gui.Form.CancelButton();
	cancel_6.setText("Cancel");
	Client.gui.Form.SubmitButton submit_7 = new Client.gui.Form.SubmitButton();
	submit_7.setText("Send");
	submit_7.setDefault(true);
	submit_7.setShell(shell);
	submit_7.setScript("connection.sendMail(to.getValue(), subject.getValue(),data.getValue())");
	form1.add(object_2);
	form1.add(input_3);
	form1.add(input_4);
	form1.add(edit_5);
	form1.add(cancel_6);
	form1.add(submit_7);
	window_1.add(form1);
	root.add(window_1);

Database Summary (Aug. 2014)

Tables

- acctperiod
- act_rules
- act_types
- activity
- advisor_type
- advisors
- area_reserve
- areas
- billaddresses
- boolean
- buddies
- buddies_messages
- calendar
- charge_classes
- charge_rules
- college
- comments
- company
- debug
- departments
- dependencies
- equip_devel
- equip_move
- equipment
- equipstats
- facilities
- flexfields
- funds
- groups
- history
- hosts
- inven_items
- inven_types
- inventory
- journal
- journal_rules
- ledger
- lineages
- locations
- mail_conf
- mask_request
- mask_request_layers
- member_groups
- member_pict
- members
- members_status
- messages
- migrate
- new_department
- objects
- onlineanswers
- onlinechoices
- onlinequestions
- onlinetest_type
- onlinetesthosts
- onlinetests
- overcapfee
- parameter_history
- parameters
- period
- periodic
- pr_funds
- pr_members
- prgroup
- problems
- problemstats
- proc_mod
- process_monitoring
- process_monitoring_data
- process_monitoring_equipments
- process_monitoring_field_groups
- process_monitoring_fields
- process_monitoring_files
- process_monitoring_photos
- process_monitoring_reports
- projects
- properties
- purchfillins
- purchforms
- purchitems
- purchorders
- purchtypes
- qualification_rules
- qualify
- qualify_history
- questionresults
- rates
- recognitions
- rep_types
- report_hist
- report_params
- reports
- res_notes
- res_procedures
- res_types
- research
- research_focus
- reservation_rules
- reserve
- resource_groups
- resource_manuals
- resource_problems
- resource_procedures
- resources
- safety_incidents
- sequences
- server_session
- session_archive
- sessions
- shipaddresses
- shipinstructs
- shipmethods
- staff
- staffrate
- suggestions
- surcharge_exclusion
- surcharge_rules
- suspensions
- symptoms_mail
- synchronize
- tasks
- taxrate
- testresults
- training
- units
- university
- us_states
- util_oprs
- util_types
- utilities
- vendors
- visitors
- voice_messages

Procedures

```
acctperiod_insert_proc
act_type_delete_proc
act_type_insert_proc
act_type_update_proc
activity_close_proc
activity_insert_proc
activity_journal_proc
activity_oc_proc
advisor_type_delete_proc
advisors_delete_proc
advisors_insert_proc
ajust_group
area_reserve_insert_proc
areas_insert_proc
billaddresses_insert_proc
calendar_insert_proc
charge_classes_insert_proc
charge_rules_insert_proc
charge_rules_update_proc
check_access_fee
college_delete_proc
comment_templates_insert_proc
comments_insert_proc
company_delete_proc
debug_insert_proc
debug_msg_proc
departments_delete_proc
departments_insert_proc
dependencies_insert_proc
dependent_report_update_proc
equip_devel_insert_proc
equip_downtime_proc
equip_downtime_proc_t
equipment_delete_proc
equipment_insert_proc
equipment_insert_proc_noreturn
equipment_update_proc
equipstats_insert_proc
equipstats_update_proc
facilities_delete_proc
facility_insert_proc
facility_update_proc
filter_by_facility
flexfields_insert_proc
funds_delete_proc
funds_insert_proc
funds_update_proc
get_ml_number
get_parent_object
get_periods
get_price
get_price_all
get_resource_location
groups_delete_proc
groups_insert_proc
history_insert_proc
history_insert_proc2
hosts_insert_proc
inven_insert_proc
inven_item_delete_proc
inven_item_insert_proc
inven_item_update_proc
inven_items_insert_proc
inven_types_insert_proc
journal_insert_proc
journal_rules_insert_proc
ledger_insert_proc
ledger_update_proc
location_insert_proc
location_update_proc
locations_delete_proc
membergroups_insert_proc
membergroups_update_proc
members_insert_proc
members_status_insert_proc
members_update_proc
messages_insert_proc
new_department_delete_proc
new_report
new_report2
next_id
onlineanswers_insert_proc
onlinechoices_delete_proc
onlinechoices_insert_proc
onlinechoices_update_proc
onlinequestions_delete_proc
onlinequestions_insert_proc
onlinequestions_update_proc
onlinetests_delete_proc
onlinetests_insert_proc
onlinetests_update_proc
parameter_history_insert_proc
parameters_insert_proc
period_insert_proc
populate_location
populate_utility
pr_funds_delete_proc
pr_funds_insert_proc
```

```
pr_members_delete_proc
pr_members_insert_proc
problem_insert_proc
problem_update_proc
problemstats_insert_proc
problemstats_update_proc
proc_mod_delete_proc
proc_mod_insert_proc
proc_mod_update_proc
projects_delete_proc
projects_insert_proc
projects_update_proc
prop_insert_proc
purchfillins_insert_proc
purchforms_insert_proc
purchitems_insert_proc
purchorders_insert_proc
purchtypes_insert_proc
qualification_rules_before_insert_proc
qualification_rules_delete_proc
qualification_rules_insert_proc
qualify_extend_by_name_proc
qualify_extend_proc
qualify_extend_rules_proc
qualify_history_insert_proc
qualify_history_insert_proc2
qualify_insert_proc
qualify_proc
questionresults_delete_proc
questionresults_insert_proc
recognitions_insert_proc
recognitions_update_proc
report2_insert_proc
report_update_proc
res_notes_insert_proc
res_notes_update_proc
res_procedures_insert_proc
res_procedures_update_proc
res_rules_before_insert_proc
res_rules_delete_proc
res_type_insert_proc
res_type_update_proc
research_delete_proc
research_focus_delete_proc
research_insert_proc
reservation_rules_insert_proc
reserve_delete_proc
reserve_insert_proc
resource_manuals_update_proc
resource_problems_insert_proc
resource_procedures_update_proc
resource_update_price_proc
resourcegroups_delete_proc
resourcegroups_insert_proc
resourcegroups_update_proc
resources_delete_proc
resources_insert_proc
safety_insert_proc
server_session_insert_proc
server_session_proc
sessions_insert_proc
shipaddresses_insert_proc
shipinstructs_insert_proc
shipmethods_insert_proc
staff_delete_proc
staff_insert_proc
staff_proc
staff_remove_proc
suggestions_insert_proc
suggestions_update_proc
surcharge_apply_proc
surcharge_exclusion_delete_proc
surcharge_rules_delete_proc
surcharge_rules_insert_proc
suspensions_insert_proc
tasks_insert_proc
testresults_delete_proc
testresults_insert_proc
testresults_submit_proc
testresults_update_proc
training_insert_proc
university_delete_proc
us_states_insert_proc
util_oprs_insert_proc
util_types_insert_proc
utilities_delete_proc
utilities_insert_proc
utilities_update_proc
vendors_insert_proc
visitor_insert_proc
```

Triggers

```
acctperiod_insert_rule
act_type_delete_rule
act_type_insert_rule
act_type_update_rule
activity_close_rule
activity_insert_rule
activity_insert_rule
advisor_delete_rule
advisor_type_delete_rule
advisors_insert_rule
area_reserve_rule
areas_insert_rule
billaddresses_insert_rule
calendar_insert_rule
charge_classes_insert_rule
charge_rules_insert_rule
charge_rules_update_rule
charge_rules_update_rule
college_delete_rule
comments_insert_rule
company_delete_rule
debug_insert_rule
departments_delete_rule
departments_insert_rule
dependencies_insert_rule
equip_devel_insert_rule
equipment_delete_rule
equipstats_insert_rule
facilities_delete_rule
flexfields_insert_rule
funds_coa_rule
funds_delete_rule
funds_insert_rule
funds_update_rule
groups_delete_rule
groups_insert_rule
groups_update_rule
history_insert_rule
hosts_insert_rule
inven_insert_rule
inven_item_delete_rule
inven_items_insert_rule
inven_types_insert_rule
journal_insert_rule
journal_rules_insert_rule
ledger_insert_rule
locations_delete_rule
membergroups_insert_rule
membergroups_update_rule
members_insert_rule
members_status_insert_rule
members_status_rule
members_status_rule
members_update_rule
messages_insert_rule
new_department_delete_rule
onlinechoices_delete_rule
onlinechoices_insert_rule
onlinechoices_update_rule
onlinechoices_update_rule
onlinechoices_update_rule
onlinequestions_delete_rule
onlinequestions_insert_rule
onlinequestions_update_rule
onlinequestions_update_rule
onlinequestions_update_rule
onlinetests_delete_rule
onlinetests_insert_rule
onlinetests_update_rule
onlinetests_update_rule
onlinetests_update_rule
onlinetests_update_rule
onlinetests_update_rule
parameter_history_insert_rule
parameters_insert_rule
period_insert_rule
pr_funds_delete_rule
pr_funds_insert_rule
pr_members_delete_rule
pr_members_insert_rule
problem_insert_rule
problem_update_rule
problemstats_insert_rule
proc_mod_delete_rule
projects_insert_rule
projects_update_rule
prop_insert_rule
purchfillins_insert_rule
purchforms_insert_rule
purchitems_insert_rule
purchorders_insert_rule
purchtypes_insert_rule
qualification_rules_before_insert_rule
qualification_rules_delete_rule
qualification_rules_insert_rule
qualify_delete_rule
qualify_history_insert_rule
qualify_insert_rule
qualify_insert_rule2
qualify_update_rule
qualify_update_rule
questionresults_delete_rule
recognitions_insert_rule
recognitions_update_rule
report2_insert_rule
report2_insert_rule
report_update_rule1
report_update_rule2
res_notes_insert_rule
```

```
res_notes_update_rule
res_procedures_insert_rule
res_procedures_update_rule
res_rules_before_insert_rule
res_rules_delete_rule
res_type_insert_rule
res_type_update_rule
research_delete_rule
research_insert_rule
reservation_rules_insert_rule
reserve_delete_rule
reserve_insert_rule
reserve_insert_rule
resource_manuals_rule
resource_price_rule
resource_price_rule
resource_problems_insert_rule
resource_procedures_insert_rule
resource_procedures_update_rule
resource_update_price_rule
resourcegroups_delete_rule
resourcegroups_insert_rule
resourcegroups_update_rule
resources_delete_rule
resources_insert_rule
safety_insert_rule
server_session_insert_rule
sessions_insert_rule
shipaddresses_insert_rule
shipinstructs_insert_rule
shipmethods_insert_rule
suggestions_insert_rule
suggestions_update_rule
surcharge_exclusion_delete_rule
surcharge_rules_delete_rule
surcharge_rules_insert_rule
suspensions_insert_rule
tasks_insert_rule
testresults_delete_rule
training_insert_rule
university_delete_rule
us_states_insert_rule
util_oprs_insert_rule
util_types_insert_rule
utilities_delete_rule
vendors_insert_rule
```