Advanced Oracle Tricks in Supporting Systems Administration

Jon Finke
Server Support Services
Rensselaer Polytechnic Institute

December 17, 1999

Abstract

Many installations (including this one) use Oracle or other relational database management systems to help manage their user account space, as well as other aspects of their operation. Over the years, we have developed a number of techniques using advanced features of Oracle to assist in this process. Since many of the people who are implementing these systems are systems administrators rather than database application developers, this paper is intended to give them some ideas of how to increase the level of automation, provide better access control and simply just explore some of the neat features and power of Oracle. This paper picks up where my earlier Oracle Tools[Fin92b] paper leaves off.

Contents

1 Introduction

Here at RPI, we have been automating many aspects of our Unix systems administration using an Oracle based package called Simon, starting with management of the actual Unix userids[Fin92a][Fin93], unix groups, printer configuration[Fin94], postmaster (`/etc/aliases`), Hostmaster[Fin92c] and many other things. We have been working on this project for over 8 years now, and have learned many things, both good and bad, about what to do and how to do it. We even have had some opportunities to redo some things based on what we have learned.

We are not alone in these efforts. I have talked with system administrators at many other sites who are working on similar projects, including the University of Alberta, Simon Frasier University, SUNY Albany, and the University of Connecticut. This is not limited to educational sites, I have also spoken with folks at Cisco Systems and Collective Technologies about similar projects. A quick glance at the last few LISA proceedings show a number of similar projects including Accountworks[Arn98], NFS Configuration Management[dSdCF+98], Unix Host Administration[TSO+96], Aurora[GMR95], Exu[RG95] (Ok, they talked about doing it) and others.

In the paper, I am going to give a very brief introduction to relational database use and simple views, and then move into complex views, stored procedures, database triggers and advanced packages, all cast in terms of systems administration. Although these examples are drawn directly from our system which is running under Oracle 8, many of the facilities are available in other databases, or similar features may be available. All of the table and PL/SQL definitions are available on the web, see the "References and Availability" section for details.

2 Starting Points and Views

The very base of our system[1] is the `LOGINS` (see table 1). This contains much of the information available in `/etc/passwd` as well as other information such as who owns the account, budget numbers, email handling, etc.

Table 1: `Logins` Oracle Table Definition

Name	Type	Size	Description
Owner_Id	Number	9	The People.Id of the person who owns this userid.
Username	Varchar2	8	The Unix username.
UnixUid	Number	16	The Unix UID for this account.
UnixGid	Number	16	The Unix group ID, if not the default.
Gecos	Varchar2	64	The Gecos or finger information field for the password file. Initially, the user's "Real name".
Source	Varchar2	16	The type of account, and it's current status such as "PRIMARY-STUDENT" or "EXPIRED-EMP".
Budget	Varchar2	32	The budget number, either a 9 digit student number or a 4 part finance ID number.
Expire_Date	Date		When this account expires (or makes the next transition).
Mail_Delivery	Varchar2	128	Optional email forward information for *Username*`@RPI.EDU`.

[1] Folks who attended the Invited Talk "Manage People, Not Userids" at LISA 10 in Chicago will know the real base is `PEOPLE` but for this example, `LOGINS` is a good starting point.

We use this table[2] to generate our `/etc/passwd` files that are pushed around via NIS and other means. You may note some missing fields - we use Kerberos for authentication, so we do not store a Unix PW crypt here (although we used to). We also don't specify a shell, that is derived from the `SOURCE` field. The home directory is generated based on the username and unixuid, so we don't need to include it here. If needed, it would be trivial to add these fields.

When we set up an account, we also set up a matching Oracle account[3]. This enables a lot of the following tricks and techniques. For the most part, our users have no idea that they have an Oracle account, and they never interact directly (via SQL) with it. There are a number of ways to handle authentication with Oracle. Initially, we did a custom network protocol to talk to the server, but many alternatives are available including Kerboros, RADIUS, SecureID and others.

This account, along with the view `My_Logins` (see figure 1), allows us to enable users to change their GECOS field in the database (and it will then be reflected out to all of the systems). In the same way, users can also change their email forwarding.[4]

Figure 1: `MY_LOGINS` view definitions

```
Create View My_Logins
    as Select Username,Gecos,Mail_Delivery,Expire_Date
  from Logins
 where Username=Substr(USER,5)
   and Substr(USER,1,4) = 'OPS\$'
  with check option;
Grant Select,update(Gecos,Mail_Delivery)
   on My_Logins to PUBLIC;
```

The `USER` value in the `My_Logins` view, is the current Oracle user. If my Username is `finkej`, my Oracle account name is `OPS$finkej`. This view is then granted to everyone, along with the right to update two of the fields. The really neat thing here, is that I can only see MY information (and change those two particular fields).

2.1 Simple Relations

The power of a relational database comes from it's ability to handle relations between data in tables. Lets look at we might handle the user's shell in the `LOGINS` table. We want the shell to base based on the type of account, which we can determine from the `LOGINS.SOURCE` column. Lets create a table called `SOURCE_TYPES`

Table 2: `Source_Types` Oracle Table Definition

Name	Type	Size	Description
Source	Varchar2	16	The type of account, and it's current status such as "PRIMARY-STUDENT" or "EXPIRED-EMP".
Shell	Varchar2	128	The shell to be used for this type of account.
Unixgid	Number	12	The default unix group id for this type of account.

We can connect the `LOGINS` table with the SOURCE_TYPES table (or *Join* them in database terms) by equating the two columns in a select

[2]The actual table definition is included in the appendix.

[3]The oracle account is of the format `OPS$`*Username*. This tells Oracle that we are using the OS authentication of the database machine for authentication.

[4]Rather than use `.forward` files, we generate an alias file for our mail server. This allows forwarding to continue after a Userid has been expired and removes the dependency on the file servers.

statement (figure 2). This will return a list of usernames, unixuids and shells for each entry in

Figure 2: Join Example

```
Select Username,Unixuid,Shell
  from Logins, Source_Types
 where Logins.Source = Source_Types.Source;
```

the LOGINS table that has a corresponding source entry in the SOURCE_TYPES table. There are a couple of things to watch out for; they may be features or may be bugs, depending on what you are trying to do. If there is no matching source value in the SOURCE_TYPES table, for a given row in the LOGINS table, that row will not be returned. There is a way to get around that called an outer join. On the other hand, if there is more than one entry in the SOURCE_TYPES table that matches, each combination will be returned.

Since you can create a view of just about anything you can select, you could create a view of the LOGINS table to give you the passwd file (figure 3).

Figure 3: ETC_PASSWD view definitions

```
Create View Etc_Passwd
    as Select Username, Pwhash, Unixuid,
        nvl(Logins.Unixgid,Source_Type.Unixgid),
Gecos, Shell, '/home/' || Username
  from Logins, Source_Types
 where Logins.Source = Source_Types.Source;
```

We do a couple of things here. We normally determine the Unix GID of an account based on the type of account. Students are in one group and staff are in another. However, we want the ability to override that group on an individual basis. So we use the NVL function to return the LOGINS.UNIXGID value if it is not null, otherwise we return the value we get from the SOURCE_TYPES table. We also do some string concatenation to build up the home directory path from the username.

2.2 Complex Views

Lets consider a more complicated problem. We want to be able to enable specific individuals within each department to do certain administrative tasks for the students within their department. We wanted to be able to designate more than one person for a department, allow someone to service more than one department, and have this happen as automatically as possible. As a starting point, we have the STUDENTS table, which is maintained by other programs based on information from the registrar (table 3).

Table 3: STUDENTS Oracle Table Definition (partial)

Name	Type	Size	Description
Person_Id	Number	12	The Person.Id of this student.
Department	Varchar2	4	The department code for this student.

To do this, we need to create a table to associate departmental administrators with departments. This gives us the Dept_Admin table (table 4)

The first step is to create a view My_Admin_Depts (figure 4) which will be all of the departments as person can work on.

Table 4: `Dept_Admin` Oracle Table Definition

Name	Type	Size	Description
Unixuid	Number	12	The Unixuid of the account being authorized to operate on students in this department.
Department	Varchar2	4	The department code for department they can maintain.

Figure 4: `My_Admin_Depts` view definitions

```
Create View My_Admin_Depts
    as Select Departments
  from My_Logins, Dept_Admins
 where My_Logins.Unixuid = Dept_Admins.Unixuid;
```

We then build on this, to create the view `My_Admin_People` (figure 5), which will be the list of all of the people who are in that department.

Figure 5: `My_Admin_People` view definitions

```
Create View My_Admin_People
    as Select Person_Id
  from Students, My_Admin_Dept
 where Students.Department = My_Admin_Dept.Department;
```

We want the departmental administrators to be able to change the email forwarding (Mail_Delivery) for their students. So, one more view of the logins table called `My_Admin_Logins` (figure 6)

Finally, we have a view that allows a departmental administrator to view and update the mail forwarding for their students. All of the access control is enforced by the database, and we don't need to rely on the application for security.

But say we want to allow things to be inserted on behalf of someone else, not updating an existing record. For the sake example, lets consider the case where we want to be able to request a billing statement be mailed to the address of record. We will start with the table `Statement_Request` (table 5).

We can give insert access to the individual via the `My_Statement_Req_Ins` view (figure 7). A couple of things to note here. In order for this view to "select" anything, not only does the person_id of the request be the current person, the time and date of the request must be the current time and date. Unless you are very quick (to the resolution of the Oracle clock), you will never get any rows out of this view. However, you can insert a new row, since at that moment in time, you will match the selection constraints and it will be allowed. We can take this concept a bit further and enable department administrators to make requests with the `My_Admin_Req_Ins` view (figure 8).

3 Stored Procedures

You may eventually need to go beyond what views can give you. You may want to apply more complex business rules (like only allow administrative password changes during business hours) or want users to be able to request more complex tasks, and provide immediate feedback.

Figure 6: `My_Admin_Logins` view definitions

```
Create View My_Admin_Logins
    as Select Username, Mail_Delivery
  from Logins
 where Owner in (Select Person_Id
                   from My_Admin_People)
  with check option;
Grant select,update(Mail_Delivery) on My_Admin_Logins to Public;
```

Table 5: `Statement_Request` Oracle Table Definition

Name	Type	Size	Description
Person_Id	Number	12	The People.Id of the person who wants a statement.
Request_Date	Date		The time and date the statement was requested.

3.1 Access Control

We currently use Kerberos for authentication for all of our user accounts (student, faculty and alumni). Users need to have their passwords reset from time to time, so we needed a way for our help desk staff to be able to reset passwords. However, we did not want our student consultants to be able to change faculty passwords. To make life even more interesting, we wanted our Alumni Relations staff to be able to change the passwords of Alumni accounts, but not any of the others. We also wanted the ability to do this without giving each of the people a Kerberos administrative account[5]

3.2 Password Change Management

Lets first come up with a way to do password changes. While we might be able to do this with something like SysCtl[DL93], we wanted some finer grain control, based on administrative switches like "student" or "alumnus". To this end, we created an oracle table to hold the requests (table 6).

Table 6: `Passwd_Change_Queue` Oracle Table Definition

Name	Type	Size	Description
Unixuid	Number	16	The Unix uid of the ID to be changed
New_Passwd	Varchar2	32	The new password encrypted with double rot-13
Request_Date	Date		The time and date when the request was made
Processed_Date	Date		The time and date when this was processed.
Process_Result	Varchar2	8	A flag indicating what happened
Clerk_Uid	Number	16	The Unix uid of the person entering the request.

Requests are put in this table, and another process running on a secure machine (with proper credentials) periodically looks in this table for entries that have not been processed (`Date_Processed` is null, made a few sanity checks, changes the password and marks it as done. (A later section will discuss how to do this in almost real time).

For our basic procedure, the only access check want to make, is to ensure that the target username is not in a list of special users, generally system administrators and folks who need special handling. To do this, we create a stored procedure, `Queue_Passwd_Change` (figure 9).

[5]Imagine root, only with more power.

Figure 7: `MY_Statement_Req_Ins` view definitions

```
Create View My_Statement_Req_Ins as
Select Person_Id, Request_Date
  from Statement_Request
 where Person_Id in (Select Owner
                       From My_Logins)
   and Request_Date = Sysdate
  with Check Option;
Grant Select,Insert on My_Statement_Req_Ins to Public;
```

Figure 8: `My_Admin_Req_Ins` view definitions

```
Create View My_Admin_Req_Ins as
Select Person_Id, Request_Date
  from Statement_Request
 where Person_Id in (Select Owner
                       From My_Admin_People)
   and Request_Date = Sysdate
  with Check Option;
Grant Select,Insert on My_Admin_Req_Ins to public;
```

Now that we have a way for the full time help desk staff to make requests, we would like to allow the alumni relations staff to change the passwords on alumni accounts. To do this, we create a second stored procedure, `Queue_Alumni_Passwd_Change` (figure 10).

This allows anyone who has been granted access to the role `Simon_Req_Alum_Pw_Change` the ability to request a new password for an alumni account. If they try to request a change for some other type of account, the request will be rejected. In a similar way, we can set up a stored procedure for our student employees to change student passwords. When an account changes from student to alumnus[6], the set of people who can "administer" it changes automatically.

4 Oracle Signals and Pipes

Oracle includes a number of packages that can be used in program development. One of these is called `DBMS_ALERT`[ABF+92] which allows Oracle applications to register interest in a particular signal, wait for them (with optional timeouts) and signal other waiting applications. Since all the signal processing is taking place on the database server, any Oracle application on any platform (that support Oracle of course) can wait for or signal processes on other machines.

If we look back to our earlier example of the password change queue, we make a change to the PW change daemon. Now after it starts up and processes any outstanding requests, instead of exiting, it registers for a signal and then starts waiting for it. When the wait terminates, (without an error status of course), it checks for outstanding requests, processes them and goes back into the loop again. We also add the following lines to the `Queue_Passwd_Change` procedure:

```
--  Signal any waiters
--
        dbms_alert.signal('PASSWORD_CHANGE_PENDING',NULL);
```

[6]This change is primary an administrative change, the account name, files and password remain the same

Figure 9: `Queue_Passwd_Change` Procedure definition

```
procedure queue_change( target_unixuid  IN Passwd_Change_Queue.Unixuid%Type,
                        target_username IN Logins.Username%Type,
                        new_passwd      IN Passwd_Change_Queue.New_Passwd%Type,
                        target_disable  IN Logins.Disabled%Type,
                        RetVal          OUT Passwd_Change_Queue.Process_Result%Type)
is
        rows            Number;
BEGIN
--
-- check to see if the username/uid is on the reject list.
        Select count(*),max(nvl(Reason,'ExclList'))
          into Rows,RetVal
          from passwd_change_exceptions PCE
         where ( PCE.Unixuid = Target_Unixuid
                 or PCE.Username = Target_Username )
           and when_marked_for_delete is null
        if ( Rows > 0 )
        then
                return;
        end if;
--
-- We passed the test, insert the record
--
        Insert into passwd_change_queue
                (unixuid, new_passwd, request_date, clerk_id)
        values (target_unixuid, new_passwd, sysdate, user);
        Retval := 'InsertOk';
end Queue_Passwd_Change;
```

If there is a password changing daemon running, it will be signaled and the password will be changed, generally in under a second. If there isn't a daemon running, the change request will be queued until a daemon is started and then it will be processed. What makes this even easier, is that since the procedure is stored in the database, we were able to add this functionality to all of the password changing programs without modifying their source code, or even re-compiling them. We just updated the stored procedure and all the applications started using them.

4.1 Package Access Control

Some of the packages supplied with Oracle, and possibly some of the ones you might write may be more powerful than you might like to release to the general users. For example, the dbms_alert package is not generally available. Rather than granting access to everyone, or even specific developers, you can instead wrap the routines to provide a more restrictive operating environment like with the `PWChange_Wait_Signal` procedure (figure 11).

In this way, we can grant execute privileges to the `PWChange_Wait_Signal` routine, and those users (or roles) can only wait for that particular signal.

5 Database Triggers

Database triggers are a very powerful tool. They let you set up a stored procedure that will be executed when anyone inserts, deletes or changes a row in a given table. Since these are part of the central database, you do not need to change applications to call these procedures, and in fact, the external applications can not bypass these triggers.

The GECOS change was one of the earliest Simon applications. Part of our normal daily procedures was to regenerate the password file and update it on disk and in the NIS maps. Last year, we started providing `/etc/passwd` services to some of our machines via DCE. Unlike the NIS

Figure 10: `Queue_Alumni_Passwd_Change` Procedure definition

```
Procedure queue_alumni_passwd_change
(       target_unixuid  IN Passwd_Change_Queue.Unixuid\%Type,
        new_passwd      IN Passwd_Change_Queue.New_Passwd\%Type,
        RetVal          OUT Passwd_Change_Queue.Process_Result\%Type)
is
        target_source   Logins.Source\%Type;
        target_username Logins.Username\%Type;
BEGIN
        Select Username,Source
          into Target_Username,Target_source
          from Logins
         where unixuid=target_unixuid;

        if target_source != 'PRIMARY-ALUMNI'
        then
                RetVal := 'NonAlum';
                return;
        end if;
--
--      it appears to be an alumnus, pass this down.
        queue_passwd_change(target_unixuid, target_username, new_passwd,
                            '', RetVal);
        return;
end queue_alumni_passwd_change;
grant execute on queue_alumni_passwd_change
   to Simon_Req_Alum_Pw_Change;
```

Figure 11: `PWChange_Wait_Signal` procedure definitions

```
procedure pwchange_wait_signal(  Message         OUT     varchar2,
                         Status          OUT     integer,
                         Timeout         IN      number)
IS
BEGIN
        dbms_alert.waitone('PASSWORD_CHANGE_PENDING',Message, Status, Timeout);
end Wait_Signal;
```

passwd image, the DCE registry started with a snapshot of our password file and then needed to be kept in sync with Simon. While we were able to modify our account creation and expiration processing to queue requests to create and expire user accounts, this did not handle GECOS changes. So, we created the trigger `Logins_Update` (figure 12). This trigger would be fired whenever the Gecos or Source field in the `LOGINS` table was changed in some way. If the the GECOS field was changed, it would call a stored procedure that handles Gecos changes (and queues a request to update things in DCE). In the same way, it looks for changes in the source field (account type) and calls another procedure to take appropriate action.

6 Conclusions

An Oracle (or other relational) database can be a very powerful tool in administering systems. You can do many things with just the basic tools (queries, simple joins, etc). However, if you start using some of the more advanced features, you can do some really amazing things with fine grained access control, enforcement of business rules, change tracking. These facilities are well worth exploring.

Figure 12: `Logins_Update` trigger definitions

```
create or replace trigger logins_update
before update of gecos,source
    on logins
   for each row
declare
begin
--
-- Look for Gecos changes
if :new.GECOS != :old.GECOS
then
        Login_triggers.Gecos_Change(:Old.Username, :new.Gecos);
end if; -- GECOS
-- Source changes
if :new.source != :old.source
then
        Login_triggers.Source_Change(:Old.Unixuid, :Old.Source, :New.Source);
end if;
end;
```

7 References and Availability

All source code for the Simon system is available on the web (or via AFS). See

`http://www.rpi.edu/campus/rpi/simon/README.simon`

for details. In addition, all of the Oracle table definitions as well as PL/SQL package source are available at

`http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html`

A number of papers on the Simon system, as well as the slides to go with this paper are available at

`http://www.rpi.edu/~finkej/Papers`

References

[ABF+92] Eric Armstrong, Steve Bobrowski, John Frazzini, Brian Linden, and Maria Pratt. *Oracle 7 Server Application Developer's Guide*, chapter Appendix A, pages A15–A20. Oracle Corporation, Dec 1992.

[Arn98] Bob Arnold. Accountworks: User create account on sql, notes, nt and unix. In *The Twelfth Systems Administration Conference (LISA 98) Proceedings*, pages 49–61. Sybase Inc, USENIX, December 1998. Boston, MA.

[DL93] Salvator DeSimone and Christine Lombardi. Sysctl: A distributed systems control package. In *USENIX Systems Administration (LISA VII) Conference Proceedings*, pages 131–144. IBM TJ Watson Research Center, USENIX, November 1993. Monterey, CA.

[dSdCF+98] Fabio Q. B. da Silva, Juliana Silva da Cunha, Danielle M. Franklin, Luciana S. Varejao, and Rosalie Belian. An nfs configuration management system and its underlying

object-oriented model. In *The Twelfth Systems Administration Conference (LISA 98) Proceedings*, pages 121–130. Federal University of Pernambuco, USENIX, December 1998. Boston, MA.

[Fin92a] Jon Finke. Automated userid management. In *Proceedings of Community Workshop '92*, Troy, NY, June 1992. Rensselear Polytechnic Institute. Paper 3-5.

[Fin92b] Jon Finke. Oracle tools. In *Proceedings of Community Workshop '92*, Troy, NY, June 1992. Rensselear Polytechnic Institute. Paper 3-1.

[Fin92c] Jon Finke. Simon system management: Hostmaster and beyond. In *Proceedings of Community Workshop '92*, Troy, NY, June 1992. Rensselear Polytechnic Institute. Paper 3-7.

[Fin93] Jon Finke. Relational database + automated sysadmin = simon. Invited Talk, July 1993. Sun Users Group - East Conference, Boston, MA.

[Fin94] Jon Finke. Automating printing configuration. In *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 175–184. Rensselaer Polytechnic Institute, USENIX, September 1994. San Diego, CA.

[GMR95] Xev Gittler, W. Phillip Moore, and J. Rambhasker. Morgan stanley's aurora system: Designing a next generation global production unix environment. In *Ninth Systems Administration Conference (LISA '95*, pages 47–58. Morgan Stanley, USENIX, September 1995. Monterey, CA.

[RG95] Karl Ramm and Michael Grubb. Exu - a system for secure delegation of authority on an insecure network. In *Ninth Systems Administration Conference (LISA '95*, pages 89–93. Duke University, USENIX, September 1995. Monterey, CA.

[TSO+96] Gregory S. Thomas, James O. Schroeder, Merrilee E. Orcutt, Desiree C. Johnson, Jeffrey T. Simmelink, and John P. Moore. Unix host administration in a heterogeneous distributed computing environment. In *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pages 43–50. Pacific Northwest National Laboratory, USENIX, October 1996. Chicago, IL.

8 Appendix 1 - Tables and Programs

8.1 GROUP_MEMBERS Table Layout

Column	Type	Notes
unixuid	number(22)	The unix uid (as found in simon.logins) of the member. This provides a degree o f isolation from username changes
group_index	number(22)	The group_index from the groups table. This is used as a join operator in orde r to link members to groups
when_inserted	number(22)	A simon.transcount of when this member was inserted into this group. May be use ful in setting "when_members_updated" in the groups file.
when_marked_for_delete	number(22)	A simon.transcount of when this member was removed from this group. Members wi th this value not null, should not be included in this group

8.2 GROUPS Table Layout

Defines unix groups, and some of the attributes of that group. Members of the groups are listed in a different table.

Column	Type	Notes
group_name	varchar2(32)	The name of the group. This name goes in the first field of the group file.
group_id	number(22)	The gid part of the group entry. This is the third field of the group file ent ry.
group_passwd	varchar2(32)	The passwd field in the group file. This is a text string. If null, it will b e inserted into the group file as a "*".
group_index	number(22)	An arbitrary identifier for this group entry. It is obtained from the sequence group_index_seq and will be used as the join key for the group members.
expire_date	date(7)	Date on which this group is to expire. May be null for most groups
update_date	date(7)	
when_inserted	number(22)	A simon.transcount value for when this entry was inserted into the group table.
when_updated	number(22)	A simon.transcount value for when something in this group definition was change d. Does not include changes in membership.
when_members_updated	number(22)	A simon.transcount value for when a new member is added to this group. This has to be maintained by the application that adds or deletes members.
when_marked_for_delete	number(22)	A simon.transcount value for when this group is marked for deletetion. Group ge nerations programs should not select groups where this is set.
platform	varchar2(24)	A platform type or wild card match (using "like") to have group entries for a s pecific platform or group of platforms. If null, group applies to all platforms .
hostname	varchar2(64)	A hostname or wild card match (using "like") to have this group only apply to a specific host or group of hosts. May become obsolete once better host support is in place in Simon.
comments	varchar2(240)	Just a space for administrators to discuss the current entry

8.3 LOGINS Table Layout

This table is the record of all current and expired (but still reserved) RCS usernames. It provides the base information for the /etc/passwd file and is used in userid generation and management. Once a username no longer needs to be reserved, it is generally moved to the "old_logins" table.

Column	Type	Notes
username	varchar2(8)	This is the actual RCS username that is assigned or reserved. In general opera- tion, usernames should be unique, however it is possible to have several reserved usernames that are the same (if they are reserved for two different purposes.)
source	varchar2(20)	This identifies the type of username (or reserved entry). This value may be changed by expiration processing, and is used to dindicate ome attributes of the username.
unixuid	number(22)	UNIX UID; should usually be between 1000 and 32767; NULL if not UNIX
pwhash	varchar2(13)	UNIX-style hash of current password, if known; If no password is needed, put " OPEN" here explicitly.
public_personal_info	varchar2(240)	Public personal information as it should appear in system list (aka gecos)
initialpw	varchar2(8)	Cleartext of initial random password, or password when reset by administrator. (Should turn into DES-encrypted RAW field at some point.)
owner	number(22)	People.id of the person currently responsible for use; usually the only person who knows the password
budget	number(22)	RPI budget number to be charged for resources used when not re-directed (either 6 or 9 digits)
mail_delivery	varchar2(80)	Mail delivery code – a forwarding address, or special command and parameters. NULL means normal local delivery. Special commands: "reject explanation_file "
disabled	varchar2(80)	NULL if okay, otherwise filename of rejection message – usually manually set
comments	varchar2(240)	Arbitrary comments on this, by human administrator for other administrators
when_inserted	number(22)	Transaction number at the point when this row was inserted
when_updated	number(22)	Transaction number at the point when this row was last changed
when_marked_for_delete	number(22)	When non-null, this is the transaction number when this row became obsolete. I t must not be deleted until all derived data have learned of the deletion.
expire_date	date(7)	When this login is intended to expire or change to the next state
unixgid	number(22)	The Unix Group ID for the username, if different from the default. Most userna mes will take the default
ppi_change_date	date(7)	The date on when the user (or sys admin) set the Public_Personal_Info field. T his is used as a single to the logins program to not update PPI from the people table.
pw_change_date	date(7)	The date on when the user last set the pwhash field in the table. This generall y occurs when the user changes their passwd. This may eventually be used to enc ourage periodic passwd changes.
budget_str	varchar2(32)	An alpha numeric string that represents the budget number that charges incurred by this userid should be charged to be default. The actual format and contents of this will depend on the type of userid; for example a student charge number would be different from a fac/staff charge number. It is up to the billing programs to provide proper handling for these values.
prev_source	varchar2(20)	Set by a trigger to record the old value of the source field when the source field is changed. This is mostly for informational purposes.
source_change_date	date(7)	Set by a trigger, this records when the source field was changed. This can help determine when a userid expires and makes it wasy to generate a list of "recent expirations" for the email servers.

8.4 Group File Generation

```
/*****************************************************************************
 *
```

```
 * gen_groups
 *
 *  By Jon Finke, March 1992
 *
 ***************************************************************************/
#ifndef lint
static char rcsid[]="$Id: etcgroup.c,v 1.6 1996/05/16 23:13:35 finkej Exp $";
#endif

#include <stdio.h>

#include <sandbox.h>
#include <sql.h>
#include <sys/param.h>
#include "sys.h"
#include "table_defs.h"
#define MAXHOSTNAMELEN 255
#ifndef FILE_COMMENTS_OK
#define FILE_COMMENTS_OK 0
#endif

static int file_comments_ok=FILE_COMMENTS_OK;
static char *gtab=GTAB;
static char *gmtab=GMTAB;
static char *gseq=GSEQ;
static char *tcount=TCOUNT;
static char *outfile=OUTFILE;

int gethostname(char *name, int namelen);

int list_members(FILE *outfile, int gindex){
  TABLE *t;
  ROW *r;
  char currname[32]="";
  int count;

  sql("Select username,%s",alt_name);
  sql("  from %s gm,simon.logins", gmtab);
  sql(" where gm.unixuid=logins.unixuid");
  sql("   and group_index=%d", gindex);
  sql("   and gm.when_marked_for_delete is null");
  sql(" order by username,%s", alt_name);
  t=sql(";");

  for (r=t->first_row, count=0 ; r ; r=r->next, count++){
    if ( strcmp(currname,r->data[1].s)){
      strcpy(currname,r->data[1].s);
      if ( count )
fprintf(outfile,",%s", r->data[1].s);
      else
fprintf(outfile,"%s", r->data[1].s);
    }
    if ( ! r->null[2] )
      fprintf(outfile,"#%s", r->data[2].s);
  }
  sql_free(t);
  return count;
}

/***************************************************************************
 *
 * get_group_db_version
 *
 * Get the db version number (the max transcount)
 *
 ***************************************************************************/
int get_group_db_version(char *platform, char *hostname){
  int result=0;
  TABLE *t;
  ROW *r;

  sql("Select greatest(nvl(max(when_inserted),0),nvl(max(when_updated),0),");
  sql("       nvl(max(when_members_updated),0),");
  sql("       nvl(max(when_marked_for_delete),0))");
  sql("  from %s", gtab);
  sql(" where (%S like platform", platform);
  sql("         or platform is null)");
  if ( hostname ){
    sql(" and (%S like hostname", hostname);
    sql("      or hostname is null)");
```

```
  }
  t=sql(";");
  if ( r=t->first_row )
    result=r->data[1].i;
  sql_free(t);

  /*
   * Now see if the members have changed....
   */

  sql("Select greatest(nvl(max(when_inserted),0),");
  sql("                nvl(max(when_marked_for_delete),0))");
  sql("  from %s", gmtab);
  sql(" where group_index in (select group_index");
  sql("                         from %s", gtab);
  sql("                        where (%S like platform", platform);
  sql("                               or platform is null)");
  if ( hostname )
    sql("                       and %S like nvl(hostname,%S)", hostname,hostname);
  sql("                        and when_marked_for_delete is null)");
  t=sql(";");

  if ( r=t->first_row )
    if ( r->data[1].i > result )
      result=r->data[1].i;
  sql_free(t);

  /*
   * Now look in logins for changes there.
   */
  sql("Select greatest(nvl(max(when_inserted),0),");
  sql("                nvl(max(when_updated),0),");
  sql("                nvl(max(when_marked_for_delete),0))");
  sql("  from Simon.Logins");
  sql(" where unixuid");
  sql("    in (select unixuid");
  sql("          from %s", gmtab);
  sql("         where group_index");
  sql("            in (select group_index");
  sql("                  from %s", gtab);
  sql("                 where (%S like platform", platform);
  sql("                         or platform is null)");
  if ( hostname )
     sql("                and %S like nvl(hostname,%S)", hostname, hostname);
  sql("                 and when_marked_for_delete is null)");
  sql("            and when_marked_for_delete is null)");
    t=sql(";");

  if ( r=t->first_row )
    if ( r->data[1].i > result )
      result=r->data[1].i;

  sql_free(t);


  return result;
}



int usage(){
  fprintf(stderr,"usage:\t-host {hostname}\n");
  fprintf(stderr,"\t-platform {platform}\n");
}

int main(int argc, char **argv){
  TABLE *t;
  ROW *r;
  int count;
  int mem_count=0;
  int db_version;
  int file_version;
  FILE *gfile;
  char *hostname=NULL;
  char host[MAXHOSTNAMELEN];
  char *platform=SYS;
  char buff[250];

  char rsql_appname[] = "RSQL_APPNAME=EtcGroup";
  char rsql_stuff[] = "SQL_DATABASE=/@joshua.its.rpi.edu";
```

```
putenv( rsql_appname );
if (!getenv("SQL_DATABASE")) putenv(rsql_stuff);

while (parsing_options(&argc, &argv, usage)) {
  if (option("host")) hostname=next_arg();
  if (option("platform")) platform=next_arg();
}

if ( ! hostname ) {
  hostname=host;
  if ( gethostname(hostname,sizeof(host) )) {
    fprintf(stderr,"error in gethostname\n");
    return 1;
  }
}

db_version=get_group_db_version(platform,hostname);
if ( db_version == 0 ){
  fprintf(stderr,"Unable to get DB version for (%s,%s); aborting\n",
  platform,hostname);
  return 1;
}

if ( file_comments_ok )
  file_version=get_internal_file_version(NULL, outfile, NULL);
else
  file_version=get_external_file_version(outfile);

if ( file_version >= db_version ) {
  fprintf(stderr,"File version (%d) > db version (%d), skipping update.\n",
  file_version, db_version);
  return 0;
}

fprintf(stderr,"Generating group file type %s, file vers=%d, db vers=%d.\n",
platform, file_version, db_version);


sprintf(buff,"%s.new", outfile);
if ( (gfile=fopen(buff, "w")) == NULL ){
  perror(buff);
  return 1;
}

if (file_comments_ok ) {
  fprintf(gfile,"#\n");
  fprintf(gfile,"# database version %d\n", db_version);
  fprintf(gfile,"#\n");
}

sql("Select group_name,to_char(group_id),group_passwd,");     /* 1-3 */
sql("       group_index, to_char(update_date),");             /* 4-5 */
sql("       greatest(nvl(when_inserted,0), nvl(when_updated,0),"); /* 6 */
sql("          nvl(when_members_updated,0))");
sql("  from %s", gtab);
sql(" where when_marked_for_delete is null");
sql("   and (   %S like platform", platform);
sql("        or platform is null)");
sql("   and %S like nvl(hostname,%S)", hostname, hostname);
sql(" order by group_id");
t=sql(";");


for (r=t->first_row, count=0 ; r ; r=r->next, count++);
fprintf(stderr,"There are %d groups to create", count);

for ( r=t->first_row , count=0 ; r ; r=r->next, count++){
  if ( file_comments_ok )
    fprintf(gfile, "#\n# Last Updated %s. Stanza DB version=%d\n",
    r->data[5].s, r->data[6].i);

    fprintf(gfile,"%s:", r->data[1].s);

  if ( r->null[3] )
    fprintf(gfile,"*:");
  else
    fprintf(gfile,"%s:", r->data[3].s);

  if ( r->null[2] )
    fprintf(gfile,":");
```

```
    else
      fprintf(gfile,"%s:", r->data[2].s);
    mem_count += list_members(gfile,r->data[4].i);
    fprintf(gfile,"\n");
  }
  sql_free(t);
  fprintf(stderr,"\n*** Generated %d entries, with %d members. ***\n",
  count, mem_count);

  if ( fclose(gfile)){
    perror("etcgroup: close datafile");
    exit(1);
  }

  sprintf(buff,"%s.old", outfile);
  if ( rename(outfile, buff))
    perror("etcgroup: backup");

  sprintf(buff,"%s.new", outfile);
  if ( rename(buff, outfile)){
    perror("etcgroup: install");
    exit(1);
  }

  if ( file_comments_ok == 0 )
    set_external_file_version(outfile, db_version);


  fprintf(stderr,"File %s installed at version %d.\n", outfile, db_version);

  return 0;

}
```

8.5 WEBSis Package

```
REM
REM Create WEBSIS. package of things
REM
set feedback on;
set verify off;
define name=WEBSIS
create or replace package &name as
--
-- $Header$
--
-- The WebSIS package is a collection of routines to demonstrate
-- (and perhaps deploy into production) a number of web based services
-- to folks.
--
-- The basic operating mode is a cgi-bin program running on a secure web
-- server that validates the user's credentials, sets up PL/SQL environment
-- and calls the ENTRY routine with a single parameter.  The Entry routine
-- determines if it is a valid request, and then calls the appropriate procedure.
-- If it is not valid, it should generate the approprate error message via
-- the Owa utils.
--
-- We also define the SHOWPAGE routine which is used to output the generated
-- HTML.
--
-- CGI Query parameters are loaded via Add_Par routine
-- CGI Environment values are loaded via the Add_Env routine/
--
-- $Log$
--
procedure Entry(progname in varchar2);
procedure Add_Par(name in varchar2, val in varchar2);
procedure Add_Env(name in varchar2, val in varchar2);
procedure Showpage;
end;
/
grant execute on &NAME to public;

/*
 * Now create the body
 */
create or replace package body &NAME as
------------------------------------------------------------------
--
```

```
-- ADD_PAR
--
-- Just pass this on the the WEBSIS_Util routines
--
-------------------------------------------------------------
procedure Add_Par(name in varchar2, val in varchar2)
is
begin
        Websis_Utils.Add_Par(Name, Val);
end Add_Par;
-------------------------------------------------------------
--
-- ADD_ENV
--
-- Just pass this on the the WEBSIS_Util routines
--
-------------------------------------------------------------
procedure Add_Env(name in varchar2, val in varchar2)
is
begin
        Websis_Utils.Add_Env(Name, Val);
end Add_Env;
-------------------------------------------------------------
--
-- SPLASH
--
-- THis is prototype procedure - used for testing
--
-------------------------------------------------------------
procedure Splash is
        gecos        varchar2(80);
        i        number;
begin
        Websis_Utils.Sis_Intro('WebSIS@RPI');
        gecos := Websis_Utils.Get_Gecos;
        htp.centerOpen;
        htp.strong(htf.big('Welcome ' || gecos));
        htp.centerClose;
        WEBSis_Utils.Dump_Vals;
        return;

end Splash;
-------------------------------------------------------------
--
-- VOICE_MAILBOX
--
-- Print Voice_Mailbox info
--
-- Consider an order by and maybe a date range check to avoid
-- old information.
--
-------------------------------------------------------------
procedure Voice_Mailbox
is
        Cursor Get_VM is
        Select Mailbox, Password, Load_date, Banner_Student,
               Firstnames || ' ' || Lastname "NAME"
          from Simon.My_Voice_Mailbox;
        Result        Get_Vm%Rowtype;
begin


        Websis_Utils.Sis_Intro('Student Voice Mailbox Info');

        Open get_Vm;
        Fetch Get_Vm
         into Result;

        if Get_Vm%NotFound
        then
                htp.strong(htf.big('No voicemail information found.'));
                htp.p('Please contact the Telecommunicatons office in VCC 215 for assistance.');
                htp.anchor('http://www.rpi.edu/dept/tele/student/Voicemail.html','Return to voicemail instructions.');
                return;
        end if;

        htp.centerOpen;
        htp.strong(htf.big('Voice Mailbox information for ' || Result.Name ));
        htp.centerClose;
```

```
        htp.p('Your voice mail box number is ');
        htp.strong(result.mailbox);
        htp.p('and your initial password is ');
        htp.strong(result.password);
        htp.p('.  For some hints on using your voice mail box, ');
        htp.anchor('http://www.rpi.edu/dept/tele/student/Voicemail.html','click here.');

end Voice_Mailbox;

------------------------------------------------------------
--
-- HTML_TAIL
--
-- Write out the closing HTML;
--
------------------------------------------------------------
procedure Html_Tail
is
begin
        htp.hr;
        htp.br;
--        htp.fontOpen(NULL,'Verdana, Arial, Helvetica,sans-serif','-3');
        htp.tableOpen(cattributes => 'border=0 WIDTH=100%');
        htp.tableRowOpen;
        htp.tableData(htf.fontOpen(NULL,'Verdana, Arial, Helvetica,sans-serif','-3') || 'Page maintained by<br>Server Support Se
        htp.tableData(htf.fontOpen(NULL,'Verdana, Arial, Helvetica,sans-serif','-3') || 'Rensselaer Polytechnic Institute (RPI)<
        htp.tableRowClose;
        htp.tableClose;
--        htp.fontClose;
        htp.bodyClose;
        htp.htmlClose;
end Html_Tail;
------------------------------------------------------------
--
-- ENTRY
--
-- Called by the CGI BIN environment - figures out what we have
-- to do, and passes the request on.
--
------------------------------------------------------------
procedure Entry(progname in varchar2)
is
begin
        if Progname = 'sis_exec.cgi'
        then
                splash;
                Html_Tail;
        elsif Progname = 'voice_mailbox.cgi'
        then
                Voice_Mailbox;
                Html_Tail;
        elsif Progname = 'dirinfo.cgi'
        then
                Web_Dirinfo.Entry;
                Html_Tail;
        elsif Progname = 'dirinfo_select.cgi'
        then
                Web_Dirinfo.Person_Select;
                Html_Tail;
        elsif Progname = 'dir_dept.cgit'
        then
                Web_Dirinfo.Dump_People;
        elsif Progname = 'vos_release.cgi'
        then
                Web_Sysctl.request;
                Html_Tail;
        else
                Websis_Utils.Sis_Intro('RCS Info -- Error');
                htp.centerOpen;
                htp.strong(htf.big('Error "' || progname || '" is unknown'));
                htp.centerClose;
                Html_Tail;
        end if;
end Entry;

------------------------------------------------------------
--
-- SHOWPAGE
--
-- Just calls the OWA_UTIL version
```

```
--
------------------------------------------------------------
procedure Showpage is
begin
        Owa_Util.Showpage;
end Showpage;
end &Name;
```

8.6 WEBSis_Utils Package

```
REM
REM Create WEBSIS_UTILS. package of things
REM
set feedback on;
set verify off;
define name=WEBSIS_UTILS
create or replace package &name as
--
-- $Header$
--
-- The WebSIS_UTILS package is a collection of service and utility routines
-- that are used by the WebSIS package, as well as packages called by the
-- WebSIS package.   This is basically part of the WEBSIS package, only we
-- broke it into two parts to avoid OTHER packages from being called by WEBSIS
-- and CALLING WebSIS.
--
-- ADD_PAR
--        Called by WebSIS (which was called by the cgi-bin program) to set
--        up the parameters for the call.
--
-- GET_VAL
--        Given a parameter name, return the value (or null).  Called by the
--        the service routines handling queries.
--
-- GET_PAR
--        Given a value(!?), return the parameter name (or null).  This is
--        a slimy way for us to create a zillion submit buttons and figure
--        out what they want.
--
-- ADD_ENV
--        Called by WebSIS to add an "environment variable"
--
-- DUMP_VALS
--        Dump the parameter list as an HTML list
--
-- ADD_ENV
--        Called by WebSIS to add an "environment variable"
--
-- GET_GECOS
--        Returns the "GECOS" (finger name) of the current user.
--
-- COMP_VALS
--        Compares two strings.  Returns True if they are different, or if
--        one is Null and the other isn't.
--
-- CHECK_AND_SET_VALS
--        Combines Get_Val with Comp_vals
--
-- YES_NO_RADIO
--        Turns a flag value into a Yes/No Radio button
--
-- SIS_INTRO
--        Puts out a standard RCS Web service header
--
-- $Log$
--
procedure Add_Par(name in varchar2, val in varchar2);
function Get_Val(val_name in varchar2) return varchar2;
function Get_Par(val in varchar2) return varchar2;
procedure Dump_Vals;
procedure Add_Env(name in varchar2, val in varchar2);
procedure Check_And_Set_Val(ParName in varchar2,
                            Target_Val in out varchar2,
                            Changed in out boolean);
function Get_Gecos return varchar2;
function Comp_Vals(S1 in varchar2, S2 in varchar2) return boolean;
function Yes_No_Radio(cname in varchar2,        -- Flag Name
                      val in varchar2)               -- Y/N or Null
return varchar2;
```

```
procedure Sis_Intro(title in varchar2);
end &NAME;
/

/*
 * Now create the body
 */
create or replace package body &NAME as
------------------------------------------------------------
--
-- Define some variables to hold query parameters and stuff
-- This is copied from the oracle OWA package.
--
  type vc_arr is table of varchar2(2000) index by binary_integer;
  q_var_name        vc_arr;
  q_var_value        vc_arr;
  num_q_vars        number := 0;

------------------------------------------------------------
--
-- ADD_PAR
--
-- Add a query parameter to the array of values
-- We don't check for duplicates, however, the FIRST will be
-- returned when queried.
--
------------------------------------------------------------
procedure Add_Par(name in varchar2, val in varchar2)
is
begin
        num_q_vars := num_q_vars + 1;
        q_var_name(num_q_vars) := upper(name);
        q_var_value(num_q_vars) := val;
end Add_Par;
------------------------------------------------------------
--
-- GET_VAL
--
-- Given a value namem return a value or null.
--
------------------------------------------------------------
function Get_Val(val_name in varchar2)
return varchar2
is
        i        number;
begin
        for i in 1..num_q_vars
        loop
                if q_var_name(i) = upper(val_name)
                then
                        return q_var_value(i);
                end if;
        end loop;
        return null;
end Get_Val;
------------------------------------------------------------
--
-- GET_PAR
--
-- Given a value, return a matching name!
--
------------------------------------------------------------
function Get_Par(val in varchar2)
return varchar2
is
        i        number;
begin
        for i in 1..num_q_vars
        loop
                if q_var_value(i) = val
                then
                        return q_var_name(i);
                end if;
        end loop;
        return null;
end Get_Par;
------------------------------------------------------------
--
-- DUMP_VALS
--
```

```
-- Dump the parameter array
--
-------------------------------------------------------------
procedure Dump_Vals
is
        i        number;
begin
        htp.ulistOpen;
        for i in 1..num_q_vars
        loop
                htp.listItem(q_var_name(i) || '="' || q_var_value(i) || '"');
        end loop;
        htp.ulistClose;
end Dump_Vals;
-------------------------------------------------------------
--
-- ADD_ENV
--
-- Adds an environment variable to the list maintained by
-- the OWA routines.  We write directly to their space instead
-- of using their loading routine.  This lets us avoid building
-- our own array, and then needing a trigger to load things.
--
-------------------------------------------------------------
procedure Add_Env(name in varchar2, val in varchar2)
is
begin
        if Owa.num_cgi_vars is null
        then
                Owa.num_cgi_vars := 0;
        end if;
        Owa.num_cgi_vars := Owa.num_cgi_vars + 1;
        Owa.cgi_var_name(Owa.num_cgi_vars) := upper(name);
        Owa.Cgi_var_val(Owa.num_cgi_vars) := val;
end Add_Env;
-------------------------------------------------------------
--
-- GET_GECOS
--
-------------------------------------------------------------
function Get_Gecos return varchar2 is
        result varchar2(80);
begin
        Select Public_Personal_Info
          into result
          from Simon.My_Logins;
        return result;
exception
        when no_data_found
        then
                return 'Special Oracle User';
        when others
        then
                raise;
end;
-------------------------------------------------------------
--
-- COMP_VALS
--
-- Compare two strings - if they are different, or one is null
-- and the other is not, return true;
--
-------------------------------------------------------------
function Comp_Vals(S1 in varchar2,
                   S2 in varchar2)
return boolean
is
begin
        if S1 is Null
        then
                if S2 is null
                then
                        return false;        -- Both Null - call them equal
                else
                        return true;        -- S1 Null, S2 Not - is a change
                end if;
        else
                if S2 is Null
                then
                        Return true;        -- S1 Not null, S2 Null - a change
```

```
                end if;
        end if;
        if S1 = S2
        then
                Return false;
        else
                Return True;
        end if;
        -- Should never get here
end Comp_Vals;
------------------------------------------------------------
--
-- CHECK_AND_SET_VAL
--
-- Intended to help comparing form fields with existing values,
-- it will get a parameter, compare to a target value, if there
-- is a change, set the target and set an extra boolean value
-- so other processing knows that there was a change.
--
------------------------------------------------------------
procedure Check_And_Set_Val(ParName in varchar2,
                            Target_Val in out varchar2,
                            Changed in out boolean)
is
        Sub_Val                 varchar2(2000);                -- It can be big
begin
        Sub_Val := Get_Val(ParName);
        if Comp_Vals(Sub_Val,Target_Val)
        then
                Changed := True;
                Target_Val := Sub_Val;
        end if;
end Check_And_Set_Val;

------------------------------------------------------------
--
-- YES_NO_RADIO
--
-- Generates a pair of radio buttons(Yes,No) with the appropriate value
-- checked.
--
------------------------------------------------------------
function Yes_No_Radio(cname in varchar2,        -- Flag Name
                      val in varchar2)                -- Y/N or Null
return varchar2
is
begin
        if val = 'Y'
        then
                return htf.formRadio(cname,'Y','CHECKED') || 'Yes '
                        || htf.formRadio(cname,'N') || 'No';
        else
                return htf.formRadio(cname,'Y') || 'Yes '
                        || htf.formRadio(cname,'N','CHECKED') || 'No';
        end if;
end Yes_No_Radio;

------------------------------------------------------------
--
-- SIS_INTRO
--
-- Put out some standard header stuff
--
------------------------------------------------------------
procedure Sis_Intro(title in varchar2)
is
begin
        htp.htmlOpen;
        htp.headOpen;
        --
        -- Attempt to supress caching, since some pages may have
        -- secret info like passwords
        htp.meta('expires',NULL,'-1');
        htp.meta('pragma',NULL,'no-cache');
        htp.meta(Null,'GENERATOR','SISWeb V0.1');
        htp.title(title);
        /* htp.base; */
        htp.headClose;
        htp.bodyOpen(NULL,'bgcolor="#FFFFFF" link="#000000" vlink="#FF0000"');
        htp.img('cislogo.gif',
```

```
            Null, 'CIS - RCS Web Info System', Null,
            'WIDTH=472 HEIGHT=100 VSPACE=2');
        htp.br;
        htp.hr;
end Sis_Intro;
end &Name;
```