AUTOMATED USERID MANAGEMENT

Simon Management System

Jon Finke (RPI)

Abstract

One of the objectives of the Simon userid management system was to automatically manage userids for all students, faculty and staff at Rensselaer. The resulting system takes input for students from the Registrar, and for faculty and staff from Payroll. These two feeds are merged with some special cases and result in a mostly automated system that create and expire RCS userids as people change their status at Rensselaer.

While the Simon Management system handles functions other than userid management, this paper will deal with the userid management functions.

1. Introduction

The Simon userid management system is based on a number of Oracle tables that are updated by a number of C programs (and a few awk scripts). Much of the data flow is managed using transaction numbers\(^1\) between tables, and between tables and “external” databases such as `/etc/passwd` and the AFS KA server.

Simon has a `students` table, which is the “simon-view” of the current list of enrolled students. This is periodically compared to a list of students from the registrar, with the resulting differences (adds, deletes and changes) being propagated to the `people` table.

In the same way, Simon also has an `employees` table, which is the “simon-view” of the Rensselaer faculty and staff. This is periodically compared to the current employee list from payroll and the differences are propagated to the `people` table.

Despite our best intentions and hopes, there are always people who are not listed by either the Registrar or by Payroll, yet need a userid. These people range from vendors who support the system, to temporary accounts for demonstrations, to guest accounts for visiting faculty and conference participants. All of these people are entered into the `guests` table using some specialized applications and via direct sql commands via `sqlplus`\(^2\). These changes are also propagated into the `people` table.

The `people` table has an entry for every person that we know of at Rensselaer, as well as their current student, employee and guest status. In this record is a unique identifier number, known as a person id, that is used as an sql join operator between this table and most of the other simon

\(^1\) Data Propagation between Oracle Tables, Community Workshop 92

\(^2\) `sqlplus` is a program supplied by Oracle that gives a way of issues sql commands to Oracle and controlling the format of the output.
AUTOMATED USERID MANAGEMENT

tables. This number is assigned automatically when someone is inserted into the people table, but
has no existence or meaning outside of the Simon database. For any given person, this number is
never changed, as it can not be wrong.

While each person at Rensselaer gets at least one userid (two if they are both an employee and a
student), there are some cases where someone needs a secondary userid for some purpose. These are
maintained in the secondaries table. The people table and the secondaries tables are combined
to maintain the logins table.

The logins table has information about the userid, both things that appear in the passwd file such
as the Gecos information, and information used for management of the userid such as the budget
number. This provides the source for things such as /etc/passwd and part of the mail aliases
file, as well generating input files for account admin programs such as uss and the oracle account
management program, orcaleid. With the use of transaction numbers, only “new” accounts are
created, the system keeps track of what it has already created.

2. Employee Management

The employees table holds a list of all current and past employees. This is periodically compared
to the payroll table which is simply a snapshot of the current payroll listing.

Maintenance of the payroll table is quite simple. We get a flat file from payroll that has 4 fixed
length fields, and this is passed through an awk script to generate a sql command file, that fills in
each of the rows for each staff member.

We then use the employees program to compare this table, with the employees table, noting
adds, deletes and changes. The employees table is defined as follows:

2.1 Oracle Table: Employees

Matches each entry in Payroll’s employee database used for merging in changes

- person_id number ID number of the person, if they have a record in the PEOPLE table
- ssn number SSN according to the latest update from Payroll
- status char(1) Payroll status: A)ctive, I)nactive, unpaid L)eave, P)aid leave
- name char(36) Name in the uppercase format Payroll uses
- dept number Budget code of department that pays the salary
- when_inserted number Transaction number at the point when this row was inserted

3 The uss program is used to create AFS accounts and disk volumes.
4 In order to use some programs to update their information such as mail forwarding, users need
   oracle “accounts”.
5 A description of the Simon,Payroll table is in the section “Other Tables”.
6 We have been following a practice of naming the program that maintains a given table or file to be
   the same as the table or file it maintains. The employees table is maintained by the employees
   program, and the /etc/passwd file is created by the etcpasswd program.
• **when_updated number**  Transaction number at the point when this row was last changed

• **when_marked_for_delete number**  When nonnull, this is the transaction number when this row became obsolete. It must not be deleted until all derived data have learned of the deletion.

• **withdrawal_date date(7)**  When this employee is no longer listed in the Payroll database. This may be due to termination from RPI, or leave of absence, retirement or they are adjunct on the off season.

• **admin_override char(1)**  When set to Y, this person is considered an employee and keeps their account active. May be useful for adjunct and retiring faculty. Set via an external admin program.

• **admin_override_date date(7)**  The date when the admin override flag was set or cleared.

• **admin_override_reason char(240)**  This is the reason why the flag was set. It should help other administrators understand why this flag was set, such as for retiring or adjunct faculty.

All of the fields from the **payroll** table are duplicated in the **employees** table. In addition, there are other columns to help manage the data flow, and to handle special cases. There are no transaction numbers in the **payroll** table, as this table is emptied and reloaded from an outside source, so no state is maintained.

### 3. Student Management

The **students** table holds the list of all current (and recent past) students. Much like the way the **employees** table is updated, the **students** table is periodically compared to the **srs.k0** table with the **students** program and adds, deletes and changes are made to the **students** table.

The **srs.k0** table is one of a set of tables that are all loaded from data from the Registrars Student Records System. They are maintained with the **import-srs** program, which uses a data dictionary and a selection list, extracts relevant fields and loads them into the srs tables. Like the **payroll** table, these tables are emptied and reloaded and have no state.

#### 3.1 Oracle Table: Students

Matches each entry in the Registrar’s data on students used for merging in changes in Student Record System (SRS) tables

- **person_id number**  ID number of the person, if they have a record in the PEOPLE table

- **student_number number**  Student number, according to registrars data at the time

- **name char(36)**  Name in uppercase format Registrar uses

- **new_student_number number**  When the student number changes, make a new record and set this value to point to the new record

---

7 Description of the **srs.k0** table is in section “Other Tables”.

---

*Subject Area: 3  Paper: 5*
- **when_inserted** *number*  Transaction number at the point when this row was inserted

- **when_updated** *number*  Transaction number at the point when this row was last changed

- **when_marked_for_delete** *number*  When nonnull, this is the transaction number when this row became obsolete. It must not be deleted until all derived data have learned of the deletion.

- **birth_date** *date(7)*  This is the student's birthdate as listed in srs.k0.birth_date. We save this to assist in matching names to automate SS number changes. Names alone are not unique.

- **withdrawal_date** *date(7)*  The date when the student no longer appears in the SRS data feed. Some processing depends on knowing how long ago a person dropped out of the feed. This is set by the students program.

- **info_release_flag** *char(1)*  This is copied from srs.k0.info_release_flag and can be used to control how much info is released on students. (assuming we get these records at all) "N" No info released, "D" Do not list in directory.

- **current_campus** *char(2)*  This is copied from srs.k0 and is used to figure who gets accounts be default. Currently only "TR" students get accounts, but this may change.

- **current_school** *char(2)*  This is copied from srs.k0, and is the school of the most recent active term.

- **current_classification** *char(3)*  This is copied from srs.k0, and indicates standing such as Freshman, Junior, Senior, Master, etc.

- **current_primary_degree** *char(4)*  This is copied from srs.k0, and contains the primary degree the student is pursuing, such as BS, MS, PHD, etc.

- **current_primary_curric** *char(4)*  This is copied from srs.k0, and contains things like MATH, MECH, CIVL, etc.

- **admin_override** *char(1)*  Set by an system administrator, this is to specifically designate a student as eligible or ineligible for a userid. If "Y", they get one regardless of other factors, and of "N", they dont. Null in most cases.

- **admin_override_date** *date(7)*  The date when the admin_override flag was set or cleared.

- **admin_override_reason** *char(240)*  A text description of why the admin_override field was set. The program that sets the flag, should prompt for and fill in this field.

One of the fields that we maintain is the **birth_date** field. While this may seem to be a rather personal, and irrelevant bit of information to keep on the users, this is used to help detect Social Security number changes. About 100 students a year have to change their student number\(^8\). Since we use the student number to match people up in the database, if left to itself, this will result in

---

\(^8\) Some of these cases are simply typos in the SS number. The other cases are foreign students who did not have a SS number when they started at RPI, but later obtained one.
the student’s userid expiring (as the old number “dropped out”), and a new userid being created for them. This is clearly not a good thing.

The registrar periodically sends a list of students who have changed their student numbers around to a number of different departments, so that they can all update their records. This is a paper memo, sent via campus mail, that generally arrives the day after the changes appear in the SRS tables. In order to handle this, the students program does a check for changing student numbers and prints out the potential changes.

```
sql("Select name, s.student_number, k.student_number\\n` from simon.students s, srs.k0 k\\n` where s.name=k.student_name\\n` and s.student_number != k.student_number\\n` and s.birth_date = k.birth_date\\n` order by name\\n`); t=sql("\\n`");
```

This will find all the students who have the same name and birthdate, but different student numbers. These are very likely student number changes. At this point, all we do is print out the list of potential changes and let someone verify and process them by hand.

As the SS number appears in a number of Simon tables, an application was developed to assist in changing the SS number in all the tables. The change_ssn program will prompt for the old number, and the new number. It then checks to see if the new number appears in any of the Simon table. If it does, someone has to go in and unsnarl things by hand. If the new number does not appear anywhere, it changes all occurrences of the old number to the new number, and records the change in the ss_changes.

4. Guest Management

The guests table serves as a catch all for all the people who need a userid, but didn’t fall into one of the two previous categories.

One frequent use of this table, is to handle new employees and students who need their userid, but their records have not caught up with them. This is especially acute with new employees, as the payroll feed is only updated monthly, and there is often a delay between when someone starts at Rensselaer and when they appear in the payroll feed. These people are entered into the guests table using the guests program as employees (or students as appropriate), and periodically, we run a cleanup program that deletes the entry in guests once they appear in employees or students.

We also have long term “guests”, such as vendor CEs who need access to the system, and some special cases such as the instructors assigned to the ROTC detachments on campus. As these people are not employed by Rensselaer, they never appear in the payroll feed, yet they are treated as employees in other aspects of the operation.

The other case is to support “demo” accounts. Things might be cleaner to handle these in a different way, but since this is presently the only out of band way to create userids, it works. Since much of the matching in the database uses social security numbers, the guests program will assign “ss numbers” for these demo userids from a sequence of invalid ss numbers.

\[9\] Description of Simon.Ss_Changes is in section “Other Tables”.

- 5 -
4.1 Oracle Table: Guests
Anyone who belongs in the People list who is not an Employee or Student

- **person_id number**  ID number of the person, if they have a record in the PEOPLE table
- **host number**  Person number of the person responsible for this guest. Host serves as contact point if we don’t have an address for the guest.
- **reason char(240)**  Text explanation of who this person is and why they are a guest
- **when_requested date(7)**  Date when a formal request was received to create this guest entry
- **when_to_expire date(7)**  Date when this person should no longer be considered a guest
- **lastname char(24)**  Family name of guest; spaces, apostrophes and such are okay;
- **firstnames char(24)**  Given names (first, middle); if abbreviated, use a period
- **nickname char(16)**  A nickname; preferred for familiar use; for pseudonyms, use an entry in the alias table
- **prefix char(8)**  Title for name, when needed; "Sir", "Mrs."...
- **suffix char(8)**  Formal suffix to name, if any, such as "Jr.", "III"
- **ssn number**  Guests’s US Social Security number, if available
- **when_inserted number**  Transaction number at the point when this row was inserted
- **when_updated number**  Transaction number at the point when this row was last changed
- **when_marked_for_delete number**  When nonnull, this is the transaction number when this row became obsolete. It must not be deleted until all derived data have learned of the deletion.
- **student char(1)**  Indicates a student entered manually. Generally, someone who has never appeared in the SRS tables, yet needs an account before they will get into SRS. Should be cleaned out aggressively.
- **employee char(1)**  Indicates a new hire who has not yet been included in the payroll feed. May not be used much once more frequent payroll feeds are arranged. Often payroll will lag, and this will serve as a bridge. Needs cleaning.
- **guest char(1)**  Indicates that this person is a guest, perhaps a vendor or visiting conference attendee.
5. People Management

The end result of the students, employees and the guests tables is the maintenance of the people table via the people program. There should be a single row in this table for each person at Rensselaer.

5.1 Oracle Table: People

Each human being we know about; derived from Employees, Students, and Guests; some fields may be changed here, but others will be overridden when the information they are derived from changes.

- id number An internal join key, unique and unchanging for each person
- lastname char(24) Family name; spaces, apostrophes and such are okay;
- firstnames char(24) Given names (first, middle); if abbreviated, use a period
- nickname char(16) A nick name; preferred for familiar use; for pseudonyms, use an entry in the alias table
- prefix char(8) Title for name, when needed; "Sir", "Mrs."...
- suffix char(8) Formal suffix to name, if any, such as "Jr.", "III"
- student char(1) N or Y, null if unknown
- employee char(1) N or Y, null if unknown
- guest char(1) N or Y, null if unknown
- ssn number RPI ID Number, which is often a US Social Security Number
- budget number RPI charge number to bill for resources consumed by this person (6 or 9 digits); if NULL, then do not allow any charges to be incurred.
- anonymous char(1) Anonymous 'Y' if name is not to be released; 'N' or NULL otherwise
- when_inserted number Transaction number at the point when this row was inserted
- when_updated number Transaction number at the point when this row was last changed
- when_marked_for_delete number When nonnull, this is the transaction number when this row became obsolete. It must not be deleted until all derived data have learned of the deletion.
- student_uid number This is used to hold the Unixid of the PRIMARYSTU account if any. This will be inserted by the logins program when the account and will be used to reclaim a login if a student returns.
- employee_uid number This is used to hold the Unixid of the PRIMARYEMP account if any. This will be inserted at creation time by the logins program, and will be used to reclaim the login if an employee returns to RPI
• guest_uid number This is used to hold the Unixid of a guest account. We don't currently have any guest accounts, this is something that needs to be cleaned up and a guest status established in logins.

6. Logins Management
The logins table gets information from the people table, and the secondaries table, and is also updated by several user programs. One of the objectives in userid management, is to automatically create a single userid for each student and employee. There are some cases where a user may need a second userid, perhaps to handle some billing needs not accommodated by the printing software, or to assist a student employee (who would not get an employee userid) keep work related disk space distinct from academic disk space.

6.1 Oracle Table: Secondaries
Each secondary userid is listed here, merged into LOGINS along with PEOPLE

- mainuser number People.ID of the primary user of this userid
- sponsor number People.ID of the sponsor; gets renewal messages; gets billed by default
- budget number Budget number IF NOT sponsor’s
- requested_username char(8) The username that should be assigned for this userid. There is potential for a conflict with an existing username. This needs to be verified on entry.
- comments char(240) Any comments for system administrators on this entry.
- when_inserted number A simon.transcount transaction number when this record was inserted into the table
- when_updated number A simon.transcount transaction number when this record was updated.
- when_marked_for_delete number A simon.transcount transaction number when this record is ok to be deleted. Actual deletion should wait until all derived data has been cleared.
- when_requested date(7) The date when this secondary userid was requested.
- when_to_expire date(7) The date when this userid should be expired.
- unixuid number The Unix uid of the userid that is created. This is used to link this record to the logins record to process the expire date and to allow more than one secondary per user.

People who are both employee and student, get both employee and student accounts. This allows the user to better handle billing and file ownership issues, and simplifies things when they cease being a student or employee.

Since the deployment of the RCS system, changes have been made to provide alternate arrangements for disk space and for printer billing to other account numbers. The need for secondary userids may be less now, than it once was.
Jon Finke (RPI)

There is a program (secondaries) that assists with the management of secondary userids. Since each secondary userid is linked to a specific person, when that person leaves Rensselaer, their secondary userids will also expire.

6.2 Oracle Table: Logins

All past and present accounts, with numeric uids, text usernames, or whatever

- **username char(8)** 8-character identifier, also called login name, userid, login, netname, or account; key for this table, but not necessarily unique, given different source values

- **source char(20)** Where this came from, why it was made: ECSVMS, ECSVM, ITSNET-NAME, RESERVED, PRIMARY (for normal staff/student workstation userids)

- **unixuid number** UNIX UID; should usually be between 1000 and 32767; NULL if not UNIX

- **pwhash char(13)** UNIX-style hash of current password, if known; If no password is needed, put "OPEN" here explicitly.

- **public_personal_info char(240)** Public personal information as it should appear in system list (aka gecos)

- **initialpw char(8)** Cleartext of initial random password, or password when reset by administrator. (Should turn into DESencrypted RAW field at some point.)

- **owner number** People.id of the person currently responsible for use; usually the only person who knows the password

- **budget number** RPI budget number to be charged for resources used when not redirected (either 6 or 9 digits)

- **mail_delivery char(80)** Mail delivery code - a forwarding address, or special command and parameters. NULL means normal local delivery. Special commands: "reject explanation_file"

- **disabled char(80)** NULL if okay, otherwise filename of rejection message usually manually set

- **comments char(240)** Arbitrary comments on this, by human administrator for other administrators

- **when_inserted number** Transaction number at the point when this row was inserted

- **when_updated number** Transaction number at the point when this row was last changed

- **when_marked_for_delete number** When notnull, this is the transaction number when this row became obsolete. It 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
AUTOMATED USERID MANAGEMENT

- **unixgid number** The Unix Group ID for the username, if different from the default. Most usernames will take the default

- **ppl_change_date date** The date on when the user (or sys admin) set the Public_Personal_Info field. This is used as a single to the logins program to not update PPI from the people table.

The logins table is used to generate the `/etc/passwd` file that is used by all of the workstations. When a user changes their password, the passwd program connects to the ka server\(^{12}\) and changes the password there, and also connects to the database server and updates the pwhash column. This allows us to provide unix hashes in the passwd files that still need them. Not all rows in the table end up as entries in `/etc/passwd`. This is determined by the `source` field. Some names are reserved\(^{13}\), and we need them here so they are not re-assigned.

There is also a `chfn` program that allows users to update their GECOS entry in the passwd file. At this point, propagation of that change doesn’t take place until the passwd file is redistributed. This happens roughly once a day. Normally the GECOS field is set based on information in the people table, and in fact, can automatically track name changes that come through from the Registrar or from Payroll. When the Gecos information is set by the user, the `PPI_Change_Date` is set. This not only tells us when the user set their information, but serves as a flag to logins to not update the Gecos information. The user’s shell is controlled in a different way, based on the session/setup system we use.

The mail hub we use for `user@rpi.edu`, does not support afs, and so can not look in the user’s home directory for a `.forward` file. Instead, users run a `forward` program that will update the `mail_delivery` field in the logins table, and these are periodically extracted and added to the alias file on the mail machine.

The users don’t actually access the logins table directly, instead they access an oracle view called `my_logins` which limits their access to just the row associated with their userid.

### 6.3 userid creation

When a new person appears in people (more accurately, someone has a student, employee or guest flag set to “Y” that was not set that way previously, or a new entry appears in secondaries, a new entry is added to logins.

Usernames are assigned based on the first 5 characters of the users last name, plus the first character of the first name, with the addition of 1 or 2 numeric digits if there is a conflict with existing entries.

The unix uid is selected from an oracle sequence. So far, we have not had to reuse and uids, and at our current rate, we have a few more years to go before we hit 32767. I expect that the uid space may have grown by that point, so that this shouldn’t be a problem. It would be pretty trivial to modify the uid selection routine to check the table for holes and re-allocation old uids, once the original owner has left. This is a last resort option, as the uid is used as an index in some other places.

\(^{12}\) The kerberos based authentication server used by the AFS file system

\(^{13}\) Special ids such as “root”, and old names that might be reclaimed, or email aliases, or names of recently expired users that we don’t want to reissue right away
Some of the admin staff are using PC/NFS for some applications, and so they need to be in a different unix group. The logins program checks each employee creation department number against the departments table. If the row for the department has a gid specified, it is included in the logins table. Otherwise a default is used when /etc/passwd is regenerated.

The departments table is also checked for an alternate billing number to use, otherwise, the budget number from the people table is used. For students, the budget number is their SS number, which is linked into the Rensselaer Bursar and Registrar system.

A password is randomly generated and assigned with the account is created. This is stored in the initialpw field. This is the only time we store clear text of a password. Unfortunately, a number of people have not yet bothered to change their initial password.

6.4 userid expiration
Deciding when to expire a userid is a bit of a challenge. We don’t generally get informed when someone leaves Rensselaer, instead they cease to appear in the SRS or Payroll feeds (or there was an expiration date set in the guests table. When this happens, the withdrawal date is set in the appropriate table, and the people program will update the corresponding flag in people, and the logins program will then set the expiration_date in the logins table.

This date depends on the type of userid, and even on the time of year. For employees, this is normally 30 days, but for students, it is set to shortly after late registration for the following semester. This gives students some time to clear up any registration problems without losing access to their userid.

Once the expiration date for a userid has passed, the userid will undergo a transition to another state. The next state is based on the source table. This will also update the expiration date so it the userid can make a transition to the next state when required.

We presently have the following states for a userid, the time between states depends on the type of userid.

1) Active. The userid will remain in this state until something outside of Simon changes, and the expiration date is then set and then expires when the expiration date is reached.

2) Disabled. The userid remains in the password file, but the shell has been changed to /bin/expired and their home directory has been unmounted and depermissioned. The data is still kept on disk, but is not available to anyone. This allows some time for the user to complain, and if the account needs to be restored, it is quick and easy to do so. Accounts remain in this state for 60 days unless re-activated.

3) Deadfiled. The data is copied to deadfile tapes, and removed from disk. The entry is removed from the passwd file. If the user had their email forwarded, email will continue to be forwarded, otherwise it will be rejected by the mail machine as an unknown user. Accounts stay in this state for 6 months.

4) Holdover. Email forwarding is stopped. The name is reserved, and will not be reused. If the original user returns, they will get this username back. Userids stay in this state for

---

| Description of departments table in the section “Other Tables”. |
| Description of the source table is in section “Other Tables”. |
at least 6 months. Once this period is passed, the record will be saved to an archive, and removed from the active table. The userid is now available for re-use.

The expiration program depends to a great extent on instructions stored in an Oracle table. This simplifies changing procedures for handling expirations when policy changes. This is described in more detail in a paper on table driven applications.\textsuperscript{16}

7. Other Tables
There are some other tables that are used in the userid management process that may be of interest. They are described below.

7.1 Oracle Table: Departments
List (from Payroll) of all the departments and some info about them

- **budget number** Normal 6digit financial code for primary department budget
- **shortname char(40)** Payroll’s abbreviation for department name
- **altbudget number** An alternate budget number to use for charging printing and disk.
- **unixgid number** The default unixgid for employees of this department to have when their RCS userid is created. This is only used at account creation, and will be null for most departments.

7.2 Oracle Table: Payroll
An oracle version of the raw data from the payroll office. This is a snapshot of the data and has no state.

- **ssn number** The social security number of the employee
- **status char(1)** A payroll status flag indicating full time, part time, or leave of absence.
- **name char(36)** The employee name, all upper case, formatted as lastname, firstname.
- **dept number** The budget number that the employee is paid from. This may not be their department, esp in cases of endowed chairs and directors and department chairs.

7.3 Oracle Table: Source
Contains all values that can be used in the logins.source field, and includes some date information on when logins should move to the next step. Also contains shell information for /etc/passwd generation

- **source char(20)** This is the value found in the logins.source field
- **next_source char(20)** This is the source value that is set next by logins when it needs to make a transition forward.

\textsuperscript{16} “Oracle Tools” Community Workshop 92

Community Workshop 92
• **shell_id number** An index into a table of valid shells. May eventually be used by etc/passwd to decide which sources get entries in the /etc/passwd file, and which ones get special shells.

• **transition_delta number** If not null, then the number of days from the current date until the transition to the next stage. This may be updated based on sysdate and the next field

• **transition_date date(7)** If not null, the date when the next transition should take place. This is used when you want a transition on a particular date, rather than a delta from the current date.

• **comments char(240)** A short human readable description of what a given source value represents.

• **allocation number** The Allocation (free) amount that this type of login gets. This is used for billing purposes

Note: The allocation field, used to specify the disk space allocation, is not really appropriate for the source table and should be moved to a different table.

### 7.4 Oracle Table: Ss_Changes
This is a record of all SS number changes in the simon database made by the changessn program. This is the only record of the change, as the entries in people, guests, students, employees and logins do not get transaction updates.

• **oldssn number** This is the original SS number assigned to the person in question

• **newssn number** This is the new SS number that replaced the old ss number

• **when_inserted number** This is a simon transaction count when the ss number was changed. The other simon tables do not get transaction counts updated for this change.

• **insert_date date(7)** This is the time and date that the SS number was changed.

### 7.5 Oracle Table: K0
Note: The comments for the SRS tables are extracted directly from the Registrars SRS data dictionary. This is why they are in upper case and in a slightly different format.

• **student_number number** A UNIQUE NUMBER WHICH WILL BE USED TO IDENTIFY THE // STUDENT THROUGHOUT THE SYSTEM. THE SOCIAL SECURITY NUMBER WILL // BE USED IN THIS AND ALL ASSOCIATED SYSTEMS.

• **student_name char(32)** DATA ELEMENTS 100 109 // STUDENT NAME AND IDENTIFICATION // STUDENT'S FULL, LEGAL NAME. // THE NAME SHOULD BE CODED IN DIRECTORY STYLE AS FOLLOWS: // LAST NAME (SPECIAL CHARACTERS AND BLANKS PERMITTED) // COMMA // SPACE // FIRST NAME

• **previous_surname char(15)** THIS IS USED FOR MAIDEN OR PREVIOUS NAME.

• **prior_student_number number** THIS DATA ELEMENT IS USED TO CROSS-REFERENCE INTO // THE HARDCOPY FILES AND IS TYPICALLY THE STUDENT'S // PREVIOUS STUDENT NUMBER.
- AUTOMATED USERID MANAGEMENT -

- **birth_date date(7)** DATA ELEMENTS 110 129 // STUDENT BIOGRAPHIC AND DEMOGRAPHIC DATA // MONTH, DAY, AND YEAR OF BIRTH OF STUDENT. // THIS IS CODED IN THE FORM (MMDDYY).

- **sex char(1)** IDENTIFIES SEX OF STUDENT

- **marital_status char(1)** IDENTIFIES MARITAL STATUS OF STUDENT

- **citizenship char(2)** THE STUDENT'S COUNTRY OF CITIZENSHIP IF OTHER THAN U.S. // THE STANDARD COUNTRY ABBREVIATION WILL BE USED. // // NOTE...STUDENTS CODED WITH A FOREIGN COUNTRY AS CITIZEN // AND VISA CODE OF PR OR RE ARE CONSIDERED PERMANENT // RESIDENTS

- **info_release_flag char(1)** THIS FLAG IDENTIFIES RESTRICTIONS ON THE RELEASE OF INFORMATION // FOR THE STUDENT.

- **date_pdated date(7)** DATE WHEN STUDENT VERIFIED PERSONAL DATA // DATE OF BIRTH // SEX ETHNIC ORIGIN CITIZENSHIP STATE OF ORIGIN // NEW YORK COUNTY

- **fraternity_code char(4)** SOCIAL FRATERNAL ORGANIZATION WITH WHICH THE // STUDENT IS AFFILIATED.

- **advisor_id number** THE SOCIAL SECURITY NUMBER OF THE STUDENT'S ACADEMIC ADVISOR. // THIS MAY BE USED TO INTERFACE WITH AN INSTITUTION'S // FACULTY FILE.

- **military_type char(2)** THE FOLLOWING INDICATE ROTC ON TROY CAMPUS // THE FOLLOWING INDICATE MILITARY STATUS OF STUDENTS AT ROME & PLATT

- **rpc_account char(4)** THE STUDENT'S RPC ACCOUNT ID AS ASSIGNED BY THE COMPUTER // CENTER.

- **current_campus char(2)**

- **current_school char(2)** THIS SCHOOL REFLECTS THE SCHOOL OF THE MOST RECENT // ACTIVE TERM. THIS DATA ELEMENT CAN ONLY BE MAINTAINED // DIRECTLY IF A STUDENT IS NOT ACTIVE IN A TERM. OTHERWISE // USE DE # 312.

- **current_classification char(3)** THE CURRENT CLASSIFICATION OF THE STUDENT AS DERIVED // FROM THE MOST RECENT TERM. // THIS ITEM CAN ONLY BE MAINTAINED DIRECTLY IF A STUDENT // IS NOT ACTIVE IN A TERM. OTHERWISE USE DE # 313.

- **current_prim_degree char(3)** THE DEGREE PROGRAM IN WHICH THE STUDENT IS CURRENTLY // ENROLLED AS DERIVED FROM THE CURRENT TERM (DE#314). // THIS ITEM CAN ONLY BE MAINTAINED DIRECTLY IF A STUDENT // IS NOT ACTIVE IN A TERM. OTHERWISE USE DE # 314.
- current_prim_curric char(4) THE PRIMARY MAJOR WHICH THE STUDENT IS CURRENTLY PURSUING // AS DERIVED FROM THE CURRENT TERM (DE#315), // THIS ITEM CAN ONLY BE MAINTAINED DIRECTLY IF A STUDENT // IS NOT ACTIVE IN A TERM. OTHERWISE USE DE # 315.

- deg_ckout_term char(3) THE TERM IN WHICH THE STUDENT IS EXPECTED TO COMPLETE HIS // DEGREE REQUIREMENTS.

- skf_maint_date date(7) THIS FIELD IS USED TO RECORD THE DATE OF LAST MAINTENANCE // ON EACH RECORD. THE DATE IS IN THE FORMAT MMDDYY AND // IS SYSTEM MAINTAINED.