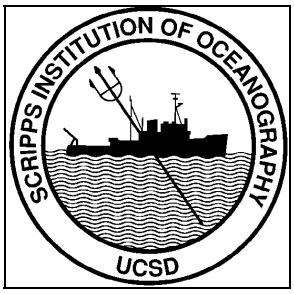

INTRODUCTORY GUIDE TO POSTGRES

Climate Research Division
Scripps Institution of Oceanography
Sequoia 2000 Project Database

DISCLAIMER:

These are my own notes taken from the POSTGRES manual and from the 2 POSTGRES introductory sessions and thus will not cover all subjects (no write-up for large-objects at this time). Refer to the POSTGRES manual and reference guide for further information. Please inform me of any mistakes in this document.

USE AT YOUR OWN RISK.

- Emelia C. Villaros-Bainto
villaros@climat.ucsd.edu

==========

o GETTING STARTED

If you don't already have one, contact Jim Frew to set-up an account (*frew@postgres.berkeley.edu*) on their system (*heel.berkeley.edu).*

o SETTING YOUR ENVIRONMENT

1) On your local workstation, add the following to your .login file:

setenv POSTGRESHOME /system/local/lib/postgres/bin

If above statement doesn't work try it without the */bin* at the end)

setenv PGHOST heel.berkeley.edu

The following must also be added to your path:

/system/local/lib/postgres/bin

OR type the following at your terminal prompt each time you use the monitor program:

set path= (/system/local/lib/postgres/bin $path)

2) In your .login file on heel:

setenv POSTGRESHOME /usr/local/postgres/bin
setenv PGHOST heel.cs.berkeley.edu

Include the following in your set path statement:

set path= (. $HOME/bin /bin /usr/bin/usr/local/postgres/bin $path)

o LOADING DATA ON HEEL DISKS

Determine the approximate amount of diskspace you will need and contact Jim Frew (*frew@postgres.berkeley.edu)* to make sure he creates a directory for you on a disk with enough space for your data.

==========

SMALL OBJECTS

==========

o PRE-PROCESSING DATA TO BE LOADED INTO DATABASE

The data to be inputted into the database can either be entered interactively by hand (one record at a time using the append command) or by using an input file. If using an input file (via the copy command), the file must follow the following rules:

1) Each record in the file must be less than 8k bytes

2) Only ASCII data can be used

3) Each field (referred to as an attribute) in the input ASCII file must be separated by tabs.

4) Each instance (record) must be terminated by a single newline

5) The file should NOT have any header fields (describing the various columns of data).

6) Embedded tabs and backslashes should be preceeded by a backslash character (\)

7) Null attributes are handled as null strings (ie. consecutive tabs in the input file denote a null attribute)

8) Array elements must be enclosed in braces, {}, and each element separated by commas. In addition, if the elements are characters, enclose them in double quotation marks.

9) The last line of the input file should have just a single period and a newline character to denote end-of-file. However, the copy command will terminate if a true EOF is encountered.

NOTE: You should know the range of values for each field in your input file so that you will know how to specify the type (i.e. int2, float4, etc) of each attribute in a class.

o CREATING YOUR OWN DATABASE

If you don't plan to share your data with other people in the *SEQUOIA* project (by placing your data in the database called *sequoia*), you will need to create your own database.

NOTE: % = UNIX prompt
* = POSTQUEL/POSTGRES prompt

Type:

% createdb databasename

- database names must have an alphabetic first character and are limited to 16 characters in length.

- you can create any number of databases at a given site and the creator will be the database administrator for each database he/she creates.

- other users can access your database but cannot make changes to it.

o USING THE MONITOR PROGRAM

The POSTGRES terminal monitor program allows you to interactively enter, edit and execute commands in the POSTQUEL query language and to access one database at a time.

Type:

% monitor databasename

You should get the following message:

Welcome to the C POSTGRES terminal monitor
Go
*

The asterisk (*) is the POSTGRES prompt indicating that it is waiting for input into the query buffer.

Here's a partial list of commands that can be typed at the POSTGRES prompt:

\e	enter the vi editor to edit the query buffer
\g	submit query buffer to POSTGRES for execution
\h	get on-line help
\i *filename*	include the file *filename* into the query buffer
\p	printcontents of the query buffer
\q	exit from the terminal monitor. this brings you back to the UNIX prompt (%).
\r	reset (clear) the query buffer
\s	escape to a UNIX subshell. To return to the terminal monitor, type "exit" at the shell prompt
\t	print current time
\w *filename*	store the query buffer to an external file *filename*
\\	produce a single backslash at the current location in the query buffer

o POSTQUEL QUERIES (Classes)

The fundamental notion in POSTGRES is that of a *class*, which is a named collection of instances of objects. Each *instance* has the same collection of named *attributes*, and each attribute is of a specific type. Furthermore, each instance has an installation-wide unique (never-changing) object identifier or *oid*.

class == table
attribute == column
instance == row

NOTE:

1) All POSTQUEL queries to be executed must be followed by a *\g*.

2) It's a good idea to start being consistent about assigning attribute names (ex. *lat* for latitude, *year* for year, etc).

3) POSTGRES is case sensitive. Use lowercase for POSTQUEL queries. Uppercase letters may be used in attribute names.

o CREATING CLASSES

A user can create a new class by specifying the class name, along with all attribute names and their types. Spaces, tabs, and newlines may be used freely in POSTQUEL queries.

Command:

** create classname (attr1 = type1, attr2 = type2,attrn = typen) \g*

where

classname	classname
attr1,..., attrn	attribute name
type1,..., typen	type of the attribute (see below)
\g	instruct POSTQUEL to execute command

Examples:

1) class with simple types

** create cnsierra (year= int4, julday= int4, precip= int4, tmax= float4, tmin= float4, tmean= float4)\g*

2) class with array of 6 values for temp

** create sst (lat= float4, long= float4, time= float8,temp = float4[6]) \g*

NOTE: Arrays are indexed starting with *array[1]* and ends with *array[n]* (similar to Fortran)

o ATTRIBUTE TYPES

(*Refer to p.22 of the reference manual for list and examples of attribute types.)*

Here's a partial list of common attribute types:

bool	boolean
char	single character
char16	fixed-length array of 16 characters (also char2,char4)
float4	single-precision floating point number
float8	double-precision floating point number
int2	2-byte signed integer (+/- 32768)
uint2	2-byte unsigned integer (+/- 65536)
int2[8]	array of 8 int2
int4	4-byte signed integer (+/- 1 billion)
text	variable-length array of printable characters

o LOADING INSTANCES INTO A CLASS

NOTE: For attributes of type text or char, use double quotes.

1) Using the append command

The user can use the append command to enter values for each attribute in an existing class. One append command for each instance in a class.

Command:

** append classname (attr1 = attr1-val, attr2 = attr2-val,..., attrn = attrn-val) \g*

Example:

a) simple example

```
* append cnsierra (year= 92, julday= 250, precip= 10,tmax= 95, tmin= 57,
                                                    tmean= 76) \g
```

b) array example

```
* append sst (lat= -9.5, long= -130.56, time= 9.010417,
                                    temp="{30.07, 30.28, 30.92, 30.79, 30.95, 30.87}")\g
```

2) Using the copy command
(see section on Pre-processing Data, above)

The copy command moves data between POSTGRES classes and standard UNIX files. The class that the data will be loaded into must already exist. There are two ways to run the copy command:

a) from the UNIX prompt (outside monitor program) or inside a script file

% monitor -c "copy classname from stdin" database < input-file

b) inside the monitor program

** copy classname from "<full pathname>input-file" \g*

NOTE: (bugs)

1) Files used as arguments to the copy command must reside on or be accessible to the database server machine by either being on local disks or a networked file system.

2) The copy command stops operation at the first error.

3) The full pathname should be used when specifying files to be copied (especially in step (2b)).

4) The copy command has virtually no error checking and a malformed input file will likely cause the backend to crash.

o ADDING ATTRIBUTES TO AN EXISTING CLASS

The initial value of each added attribute for all instances is "null". For efficiency reasons, default values for added attributes are not placed in existing instances of a class.

Command:

** addattr (attrn = typen) to classname \g*

o RENAMING A CLASS OR ATTRIBUTE IN A CLASS

The rename command will change the name of a class or attribute without changing any of the data contained in the affected class.

Command:

** rename classname1 to classname2 \g*

** rename attr1 in classname to attr2 \g*

o REPLACING VALUES OF ATTRIBUTES IN A CLASS

The replace command changes the values of the attributes specified in the target-list for all instances which satisfy a particular qualification. Only attributes that are to be modified need appear in the target-list.

Command:

** replace instance_var (attr-1=expression-1 [,attr-n=expression-n])*
[from from-list] [where qual] \g

NOTE:

1) stuff in brackets are optional.

2) *instance_var* is either a class name or a variable assigned by from-list.

Example:

** replace cnsierra(cnsierra.precip= -9.99) where cnsierra.precip= 0 \g*

- this replaces instances where precip=0 in the cnsierra class with precip= -9.99

o RETRIEVING DATA FROM A CLASS

The retrieve command will get all instances which satisfy the qualification, *qual*, compute the value of each element in the target-list, and either return them to an application program through one of two different kinds of portals or store them in a new class.

Command:

```
* retrieve  [(into classname [archive_mode] | portal portal-name) |
                iportal portal-name]  [unique]
                ([attr-1=] expression-1 {,[attr-n=] expression-n])
                [from from-list] [where qual]
                        [sort by attr-1 [using operator] {attr-n [using operator]}]\g
```

- stuff in brackets are optional.

Examples: *retrieve (classname.attribute...)*

1) To retrieve all the data in a class in table format

```
* retrieve (cnsierra.all) \g
```

2) To retrieve year and julian days where precip = 0 in class cnsierra

```
* retrieve (cnsierra.year,  cnsierra.julday) where cnsierra.precip = 0  \g
```

3) To sort what is retrieved in ascending order

```
* retrieve (cnsierra.year, cnsierra.julday) where cnsierra.precip = 0
                        sort by year using <  \g
```

NOTE: If you don't like the default table format which includes headers at the top and dashed lines to separate each instance then use (back at the UNIX prompt):

```
% monitor -T -N databasename
```

the "-T" suppresses the dashed lines
the "-N" suppresses the header lines

o RETRIEVING DATA FROM JOINS BETWEEN CLASSES

You can retrieve data from two different classes within a database.

Example:

```
* retrieve (m.long, m.lat, depth = -s.depth, s.temp)
        from m in nodc_meta_raw,
                s in nodc_bt_raw
        where s.cruise = 77506
                and m.cruise = s.cruise
                and m.station = s.station
        sort by long using <,
                lat using <,
                depth using >
        \g
```

This retrieves the longitude, latitude from a class called *nodc_meta_raw* represented by *m* and the depth and temperature from a class called *nodc_bt_raw* represented by *s* for matching cruise and station numbers between the two classes and the resulting instances will be sorted in ascending order by lat and long and in descending order by depth. The statement *depth = -s.depth* makes the retrieved depth value negative (standard notation).

o USING A SCRIPT FILE TO ISSUE MONITOR COMMANDS

Command:

```
monitor -T -c "command" database
```

Example script file contents:
(NOTE: Backslash after the quotes is a continuation mark)

```
monitor -T -c "retrieve (cnsierra.year, cnsierra.julday)
        where cnsierra.precip = 0
        sort by year using < " \
        dbname > outfile
```

NOTE: (bug)

I have had problems when I've included comments or any other lines BEFORE the command to be issued (i.e. make the first line of the script file be the *monitor* statement itself)

o OTHER RETRIEVE QUERIES FOR EXAMINING STATE OF THE DATABASE

1) To retrieve the names of all the user-defined classes (tables) in a database

```
* retrieve (pg_class.relname) where pg_class.relname !~ "^pg_.*"  \g
```

2) To retrieve the names of all classes plus additional info on each class

```
* retrieve (pg_class.all) \g
```

3) To retrieve the user-defined names and types of all the attributes for a given class ("*myclassname*")

```
* retrieve (a.attname, t.typname)
        from a in pg_attribute, t in pg_type,
                c in pg_class
        where a.attrelid = c.oid and a.atttypid = t.oid
                and a.attnum > 0
                and c.relname = "myclassname" \g
```

4) "Quick and dirty" trick to display attribute names for a given class (this statement is never true so it just displays the header line):

```
* retrieve (classname.all) where 1=0 \g
```

o DELETING INSTANCES FROM A CLASS

The delete command removes instances which satisfy the qualification, *qual,* from the class specified by *instance_var*. *Instance_var* is either a class name or a variable assigned by *from-list.* If the qualification is absent, the effect is to delete all instances in the class. The result is a valid but empty class.

Command:

```
* delete instance_var [from from-list] [where qual] \g
```

Example:

```
* delete cnsierra where cnsierra.precip < 0 \g
```

- deletes records in cnsierra class where precip is < 0.

o DESTROYING A CLASS

The destroy command removes classes from the database. Only the owner of the database may destroy a class.

NOTE: The destruction of classes is not reversible. Thus, a destroyed class will not be recovered if a transaction which destroys this class fails to commit. In addition, historical access to instances in a destroyed class is not possible.

Command:

* *destroy classname1 [,classname2,...,classnamen] \g*

o **CLEANING UP A DATABASE**

The vacuum command is the POSTGRES vacuum cleaner. It opens every class in the database, moves deleted records to the archive for archived relations, cleans out records from aborted transactions, and updates statistics in the system catalogs. The vacuum command operates on the current open database. Running a vacuum periodically will increase POSTGRES' speed in processing user queries.

Command:

* *vacuum \g*

o **WHO TO CONTACT FOR QUESTIONS**

1) Diskspace and accounts for POSTGRES

   Jim Frew *frew@postgres.berkeley.edu*

2) POSTGRES questions/tips/schema

   Support group *sequoia-support@postgres.berkeley.edu*

3) POSTGRES bugs

   Mike Olson *mao@cs.berkeley.edu*