POSTGRES Version 3
Release Notes

1991/08/18

1. Introduction

These are the release notes for version 3.0 of the POSTGRES database system from UC Berkeley. The database system and its installation procedure are covered in detail in the setup document for this release. Here, we cover only the most important differences from release 2.1 and earlier versions of the system.

2. Changes to POSTQUEL

The POSTQUEL query language has been changed to better support remote access to databases over a network, and to provide more control over data storage.

2.1. Createdb, destroydb

The *createdb* and *destroydb* commands have been added to the query language. Syntax is `createdb dbname` or `destroydb dbname`. There must be a database open while these commands are being run. We recommend running these queries from inside the "template1" database.

This change was made in order to permit users to create and destroy databases without logging onto the machine on which the postmaster is running.

2.2. Vacuum

The *vacuum* command has been added to POSTQUEL. This command causes the open database to be vacuumed. Vacuuming updates statistics on relations, cleans out records from aborted transactions, and sets commit times on committed transactions. Syntax is `vacuum`.

This query may be submitted at any time. It is a good idea to update statistics after copying in a large file, or deleting a large number of records from a class.

This change was made in order to get rid of the *vacuum* Unix program, which was very large and required maintenance separate from the POSTGRES backend.

2.3. Create syntax changes

The *create* command for creating new classes has some new optional specifications. The syntax

```
create foo (a = int4) store = "magnetic disk"
```

will cause the new class "foo" to be created, and to be stored on magnetic disk. Similarly,

```
create foo (a = int4) archive = heavy arch_store = "magnetic disk"
```

creates class "foo" as an archived class; historical data is stored on magnetic disk.

As the second example above implies, archival of user data is supported in the current release. Scans of archived data are planned and executed for historical queries, and the vacuum cleaner (see *vacuum*, above) moves data from the current store to the archive.

In the released system, the only possible location for user data is “magnetic disk”; this is also the default. The additional specifications for storage managers were added to support research being done at Berkeley.

Some caveats here: Since “store” and “arch_store” are new keywords, they may not be used in queries except as shown above. In addition, if an archived relation is destroyed, historical queries are no longer guaranteed to return the correct answer.

2.4. Portals

The command

```
retrieve portal myportal (pg_user.all)
```

in release 2.1 and earlier versions of POSTGRES could be issued as a stand-alone transaction. This violated isolation requirements, since the results from the retrieve were not actually instantiated until they were `fetch`ed, but locks were released on successful completion of the `retrieve`.

As of release 3.0, operations on portals must be done inside of `begin`/`end` transaction blocks. This change may require you to rewrite libpq application programs, if your applications use portals outside of transaction blocks. The correct interaction is now

```
begin
retrieve portal myportal (pg_user.all)
fetch 10 in myportal
      ...
close myportal
end
```

If the user fails to close a portal before the end of the transaction block, it is freed automatically, and all resources it holds are released.

3. Unix command changes

The names and behavior of some Unix commands shipped with POSTGRES have changed in the new release.

3.1. Createdb, Destroydb

The *createdb* and *destroydb* commands are now shell scripts, rather than compiled programs, and submit queries via the terminal monitor. The postmaster must be running in order for these scripts to work. For the sake of backward compatibility, we have provided two different scripts (createdb.sh and destroydb.sh) that run without using the postmaster or shared resources. Use of these two scripts is discouraged; if at all possible, use *createdb* and *destroydb* instead.

This change was made to make creating and destroying databases from remote machines easier, and to get rid of extra binaries. Since POSTQUEL now provides both of these as query language commands, special programs are no longer necessary.

3.2. Vacuum, Vcontrol

Since the query language now supports the *vacuum* command, the *vacuum* Unix program has been replaced by a shell script which submits a query via the terminal monitor. The *vctontrol* program is no longer shipped with POSTGRES. The syntax for the new script is `vacuum dbname`. This opens a connection to a backend using database “dbname” and vacuums it.

We strongly recommend that production databases be vacuumed nightly. This will keep statistics current, and permit the query optimizer to make better choices in planning user queries. The easiest way to make this happen is to add entries of the form `vacuum dbname` to your crontab file.

This change was made to get rid of the separate *vacuum* program, ease maintenance, and make the POSTGRES distribution smaller.

3.3. Monitor

The terminal monitor takes several new flags. The “−q” flag suppresses conversational output. The new “−c” flag causes the argument immediately following it to be executed as a POSTQUEL query; this overrides the normal interactive behavior of the monitor. For example,

```
monitor -c "retrieve (pg_class.all)" mydb
```

executes the query “retrieve (pg_class.all)” on “mydb”, sending the output to stdout. After the query completes, the terminal monitor exits.

The −c switch was added to make it easier to use POSTQUEL queries in shell scripts. The −q switch was added because “I live to serve you” stopped seeming cute at three o’clock one morning.

3.4. Initdb

The *initdb* shell script initializes the template database for POSTGRES. In general, it is run by the installation process, and need never be run by users. *Initdb* creates the data/ directory hierarchy in $POSTGRESHOME, sets up some control files, and populates the template database *template1*.

This change was made to get rid of the “baby backend” that handled bootstrapping of POSTGRES in previous releases. The functionality of that program was rolled into the standard backend. In addition, *initdb* substantially simplifies the installation procedure, and gets rid of confusing syntax required for *createdb* in earlier releases.

4. General System Changes

The new release includes bug fixes to many problems reported in version 2.1 and previous releases. We have also sped the code up substantially. This section contains a partial list of enhancements made since release 2.1.

4.1. Multi-User Stability

We have done extensive rewriting and testing of the code that handles concurrent access, and believe that POSTGRES is much better suited for multi-user applications than it has been in the past. The lock manager has been entirely rewritten, and exclusion is guaranteed around critical sections. Standard two-phase locking is provided on user tables. System catalogs and indices have locking strategies that provide higher concurrency.

As a new feature, if one POSTGRES backend terminates abnormally (dumps core), then all backends running concurrently are killed and restarted. This is to guarantee that any data corrupted by the errant backend are destroyed, and not flushed to disk or propagated to other backends.

We are particularly interested in reports of bugs caused by concurrent access to data by multiple users.

4.2. Storage Manager Switch

Previous releases of POSTGRES had built-in assumptions about the underlying storage medium. All data were assumed to reside on magnetic disk. Version 3.0 removes this assumption from the code. Data may reside on other media, at the discretion of the database administrator. Access is location-transparent.

At Berkeley, we have implemented a storage manager for a Sony optical disk WORM jukebox and for persistent main memory. Neither of these is supported in the released system.

4.3. Executor Rewrite

The query executor has been almost completely rewritten, in order to simplify code maintenance and to speed up execution. Several memory leaks have been plugged, and performance has been substantially improved.

4.4. Query Rewrite Rule System

The query rewrite rule system has been completely rewritten for this release. This version implements the almost all the rewrite rule semantics specified in "On Rules, Procedures, Caching, and Views" by Stonebraker et al. and closely parallels the 'instance' level rule system. It offers performance advantages over 'instance' rules in some applications.

Views, versions, and POSTQUEL functions are now supported via the rewrite system. See the 'define rule(commands)' section of the reference manual for details.

4.5. Aggregates

Aggregates (or column-valued functions) are now supported. The Reference Manual describes how to define and use aggregates in detail.

4.6. Large Objects

Large Objects are now supported as "filename as an ADT". The Reference Manual describes the large object implementation in detail. The 3.0 implementation is the first-cut implementation; a more fully-featured implementation will be available in Release 4.0.

5. Known Bugs

There are a few known bugs that we were unable to fix in the current release.

5.1. Indices and the Instance Level Rule System

The Instance Level Rule System essentially ignores indices, so if you are defining a rule on an indexed attribute, you should use the Query Rewrite rule system.

5.2. Retrieve Into and failed backends

If a backend fails while in the course of executing a Retrieve Into query, a spurious file, with the same name as the target class of the Retrieve Into, will be left in the database directory. This file can be safely deleted by the database DBA.

5.3. Large Objects and failed backends

If a backend fails while it is manipulating large objects, spurious large object files will be left in the database directory. Also, there is no mechanism for getting rid of large objects which are returned by functions but not stored in instances.

5.4. Postgres User Id's and Unix UID's

The userid of a Postgres registered user **must** match the user's UNIX user id. In the release, the user id of the Postgres user in /etc/passwd is presumed to be 6. If it is not, Postgres will not run properly. This can be overcome for those who do not wish to renumber the userid field in their password files by editing the file

src/lib/H/catalog/pg_user.h

and changing the "usesysid" field for the Postgres user to the one used in /etc/passwd at your site **before** compiling and installing Postgres. The line in this file you wish to change looks like this:

```
DATA(insert OID =    6 ( postgres 6 t t t t ));
```

If you change the two 6's above to the value used for the "postgres" user in your /etc/passwd file, this problem can be avoided.

6. Machine-dependent Problems

Postgres has been known to crash SunOS 4.0.3 on Sparcstations, due to a SunOS bug in shared memory. It appears to work on SunOS 4.1 and higher, so any reports of crashes on SunOS 4.1 and higher are appreciated.