POSTGRES Version 3.1
Release Notes

91/12/02

1. Introduction

These are the release notes for version 3.1 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, which can be found in the file ˜postgres/doc/postgres-setup.me. Here, we cover only the most important differences from release 3.0 and earlier versions of the system.

2. Aim

The main focus of this release was an attempt to fix the reported bugs in version 3.0. Because of this very little new functionality has been added to the system. One of the primary development efforts was in the push towards "ansification" of the source code. This manifested itself in the creation and clean up of POSTGRES function prototypes. All DEC platforms now compile with -DPROTOTYPES defined. In the future we intend to continue this effort in the hopes of substantially cleaning up the code.

3. Changes to POSTQUEL

The POSTQUEL query language has been modified only slightly.

3.1. define function

The syntax for defining function has been changed in a move to make functions more orthogonal. Where before there were two separate queries, one for c functions and one for postquel, now there is a single command. The language in which the function is defined must now be specified as a parameter. The syntax is as follows:

```
define function <function-name>
(language = <lang>, returntype = <typename>
    [, arch_pct = <percentage | pre-defined>]
    [, disk_pct = <percentage | pre-defined>]
    [, byte_pct = <percentage | pre-defined>]
    [, perbyte_cpu = <int | pre-defined>]
    [, percall_cpu = <int | pre-defined>]
    [, iscachable])
 arg is (<type-1> [ , <type-n>])
 as <filename or code in language as appropriate>
```

Both the demo and the video demo give examples. Currently the only languages supported are **C** and **POSTQUEL.**

4. General System Changes

4.1. Aggregates

Aggregate (or column-valued functions) support is now much more general. The type of values aggregated can be of completely arbitrary type. Due to this generalization the initial values for aggregates are now stored in the pg_aggregate system catalog as type text. This requires a slight change of syntax in aggregate definitions. The initial values must now be enclosed in double quotes (See the Reference Manual). There are also many more built-in aggregate functions. The Reference Manual describes how to define and use aggregates in detail. The demo gives some simple examples.

4.2. Functional Indices

POSTGRES now supports secondary indices defined on the resultant values of functions. The syntax for functional indices is similar to that of standard secondary indices. The difference is where you would normally specify an attribute, you now give a function call:

```
define index foo on bar using btree (f([att1] [, att2] ...) int4_ops)
```

Only attributes defined in the specified relation can be used in the function. Functional index support is restricted to cachable functions only. Defining a functional index on a non-cachable function (e.g. one that depends on current database state) can result in incorrect behavior.

4.3. Manual reorganization

The POSTGRES tutorials have been merged into a single user manual and largely rewritten. This manual is recommended reading for new users. The old manual is now called the Reference, and is still included in its entirety. The new manual is in

~postgres/doc/manual.me

5. Known Bugs

There are a few known bugs that we did not 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.

5.5. Known bugs list

A list of known bugs and suggested work-arounds can be anonymously ftp’ed. This list is kept in the file ˜ftp/pub/postgres-v3r1.bugs. We will make every attempt to keep this list up to date.

6. Machine-dependent Problems

6.1. Sparcstations running SunOS 4.0.3

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.

6.2. The vacuum cleaner on Sparcstations

The vacuum cleaner has been reported in some cases to crash the backend on Sparcstations. We have reproduced this problem, but unfortunately were unable to fix the error prior to this release going out.