# POSTGRES Version 4.1
# Release Notes

**1994/03/12**

## 1. Introduction

These are the release notes for version 4.1 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 `.../doc/postgres-setup.{cat,ps}`. Here, we highlight some of the major changes.

## 2. Aim

This release has both extensive superficial changes, such as the layout of the source tree and associated build procedures, and major new pieces of functionality.

## 3. Existing 4.0.1 Databases

There have been major changes to the tuple header structures and the system catalogs since POSTGRES Version 4.0.1. Because of this if you want to migrate your data from Version 4.0.1 you must copy all your databases out to flat ASCII files and then reload them into the 4.1 database directory. See the `copy` command in the reference manual for details about how to copy data out of a class into a flat ASCII file.

## 4. Superficial Changes

The layout of the source tree has been restructured for this release. All of the source code for the system including the reference manual and documents lies below the `src/` directory at the top level. The remainder of the directories at the top level contain a completely installed POSTGRES system suitable for use on the selected platform. Since we are only officially supporting two platforms, we thought we'd give this a shot and see if it works. In theory this allows you to get running right away without compiling the sources. If you opt to run this pre-installed system, we urge you to re-compile the system from scratch should you run into any problems. This is a test, as they say.

Underneath the `src/` directory there are subdirectories corresponding to the major components of the system. There are Makefiles everywhere for building the system. This release uses a new version of make called `bmake`; the source (and executable) is provided with the release (see `.../doc/postgres-setup.{cat,ps}`). The file `.../src/Makefile.global` is a general configuration Makefile that is included by all the other Makefiles, and has knobs and buttons to tailor this release to your specification. In general, wherever a `Makefile.global` exits, it is included by the Makefile in that directory and by any Makefiles in any subdirectories. Running `bmake` at a top level will run all Makefiles in any subdirectories too. A few exceptions to this are the `doc`, `ref`, `regress`, and `tools` directories. Rebuilding the reference manual and manpages in `doc` and `ref` requires that groff and friends be installed on your system, so we thought it best to leave these out by default. To build them you can `cd` into the directory and type `bmake install` to explicitly build them. Or you can edit `src/Makefile` to add them by default. The `regress` directory runs demos, benchmarks, and regression tests, which can potentially take hours to run -- so we thought it best to leave them out as well. The `tools` directory contains the source for the `bmake` program, which is a bootstrap piece of sorts. If you can't use the pre-compiled version of bmake, then you'll have to recompile it in here. It's all in the `postgres-setup` document.

Structurally, the `src/` directory contains the following major systems:

> `backend` - Code for the backend and postmaster,
> including all header files.
> `bin` - All the support programs, including shell
> scripts and executables.
> `contrib` - User contributed software.
> `libpq` - Frontend library code (produces libpq.a and
> copies out header files).

The `old_doc` directory contains *roff source for documents that we didn't have time to convert to our new build system, but which we thought might come in handy for reference. Should you take the time to make this work and produce Makefiles for these documents, please forward them to us.

You might also notice directories called `obj/` scattered throughout the source tree. These directories are used by bmake as a place to store the output of compilations and for temporary files. Here at Berkeley we have a special scheme that allows us to share sources with multiple hosts and platforms and build off the same source tree by having bmake do all its work in these `obj/` directories. On our master source tree these `obj/` directories are not real directories, but pointers to local storage on our workstations. When we produce the release, we turn these files back into real directories. In general, it keeps the source tree cleaner to have all the objects and executables stored down a layer in the `obj/` directory. We regret that we are not supporting this mechanism for multi-client / single source builds yet, but may do so in a future release if there is interest expressed by the user community.

## 5. New Features

### 5.1. User Authentication

This release supports user authentication using the **kerberos** system distributed by MIT. By default this system is disabled, but can be enabled by turning on the feature in `src/Makefile.global`. It requires that you have installed kerberos at your site. The kerberos code is not supplied with this package. See the file `.../doc/kerberos.faq`. Also, refer to the reference manual for details.

### 5.2. Access Control

POSTGRES provides new mechanisms to allow users to limit the access to their data that is provided to other users. See the reference manual under **ACCESS CONTROL** for details.

### 5.3. Untrusted Functions

Support has been added for designating a user-defined function as "untrusted", which means that the backend will not crash if the function fails, and that the function will run in a separate address space from the backend. See the `define function` command in the reference manual.

### 5.4. Predicate Migration Algorithm

Version 4.1 supports the complete Predicate Migration Algorithm as described in the literature [JMH92, HS93]. This means that the POSTGRES optimizer now produces optimal plans for queries that contain expensive functions in the `where` clause.

BUGS: The Predicate Migration Algorithm assumes that function results can be cached. Since function caching has not yet been implemented, the results of Predicate Migration may be suboptimal in some situations, particularly for queries that contain both expensive restrictions as well as joins that produce more tuples in the output than in the inputs (e.g. cross-product joins, joins on operators other than =, etc.)

Referrences: [JMH92] Joseph M. Hellerstein, "Predicate Migration: Optimizing Queries With Expensive Predicates", University of California, Berkeley, Tech Report Sequoia 2000 92/13, December, 1992.

[HS93] Joseph M. Hellerstein, Michael Stonebraker. "Predicate Migration: Optimizing Queries With Expensive Predicates", to appear Proc. ACM-SIGMOD International Conference On Management of Data, 1993.

### 5.5. Ubiquitous Time Travel

Version 4.1 includes an initial implementation of ubiquitous time travel for POSTGRES. Previous releases allowed the user to do time travel on data, by writing queries of the form

```
retrieve (e.all) from e in emp["Jan 1 1992"]
```

In the current release, this notion of time travel is extended to include operators and functions. For example, in POSTGRES 4.1, the query

```
retrieve (higher_sal = 1.1 * e.salary) from e in emp["Jan 1 1992"]
```

will use the multiplication (`*`) operator that existed as of Jan 1, 1992. The operator is chosen to match the time qualification applied to the data on which it operates.

Multi-table queries in which all of the tables have different time qualifications make it difficult to choose the time that should be used when looking up operators or functions. Because of this, time travel on operators and functions is only applied to queries on single tables. This restriction will be removed in a subsequent release of POSTGRES.

### 5.6. Partial Indexes

POSTGRES now supports partial indexes, which index only those instances in a class that satisfy a specified predicate. A partial index can be extended to cover more instances or to become a complete index. Partial indexes cannot satisfy as wide a range of queries as complete indexes, but they can be built more quickly, especially if the index key is an expensive function. See the `define index(commands)` and `extend index(commands)` sections of the reference manual for more details.

## 6. Not Yet But Soon

### 6.1. Sets

The implementation of sets planned for the 4.1 release of Postgres is not yet stable enough to distribute. It will be completed and released separately within approximately 1-2 months. Initially, sets will contain tuples produced as the result of some query. The user will define a set by specifying the query which produces the tuples she wants in the set. A more complete description will be provided when the implementation of sets is released.

### 6.2. Function Prototypes

We plan to produce function prototypes for the entire source tree for the next release. We decided that we will completely convert without providing backward support for traditional C because all the mechanisms that provide backward support are ugly beyond description (and still have problems with narrow argument passing). If you feel that you will be seriously impacted by this -- we would like to hear from you.