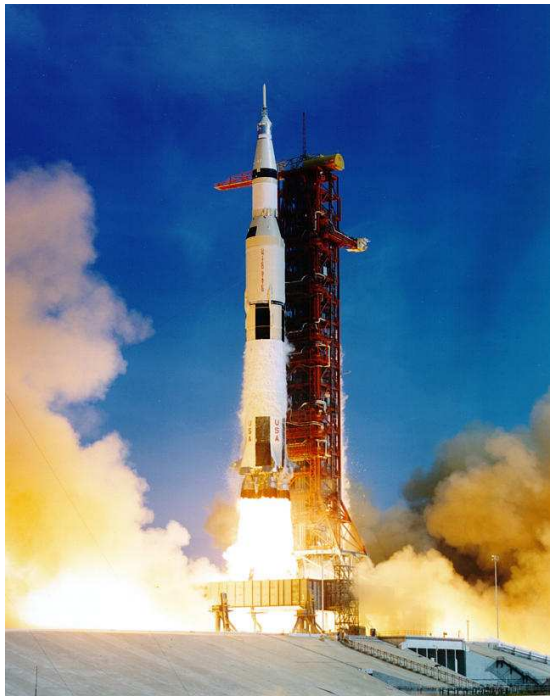
# Great Steps In PostgreSQL History

BRUCE MOMJIAN,
ENTERPRISEDB

May, 2007

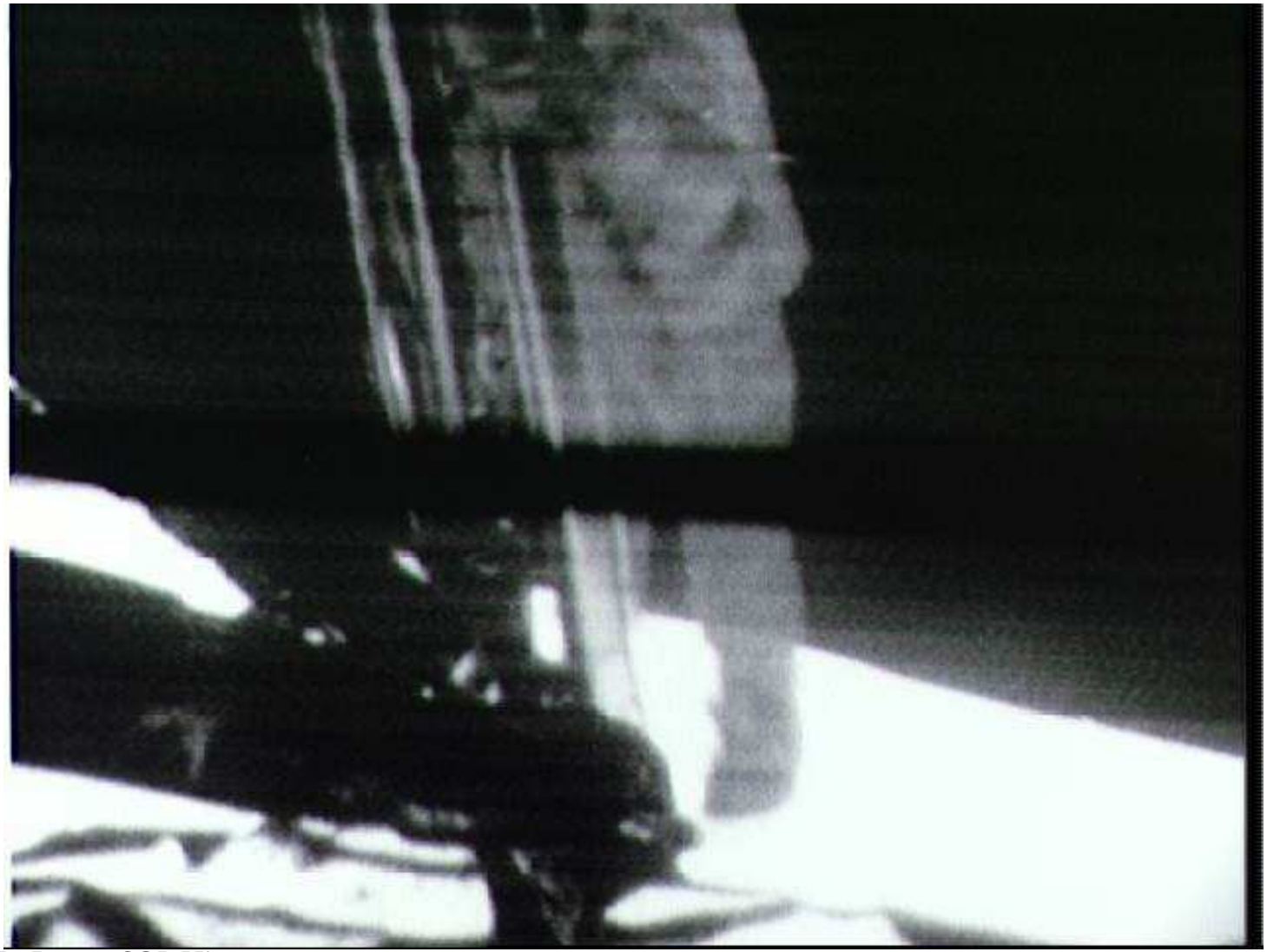

**Abstract**

This talk is about the great accomplishments in PostgreSQL history, and how they didn't seem so great at the time.



# *Apollo 11 Moon Landing: One of the Greatest Technical Accomplishments of the 20th Century*

# First Steps



## First Words

*That's one small step for man*
*One giant leap for mankind.*

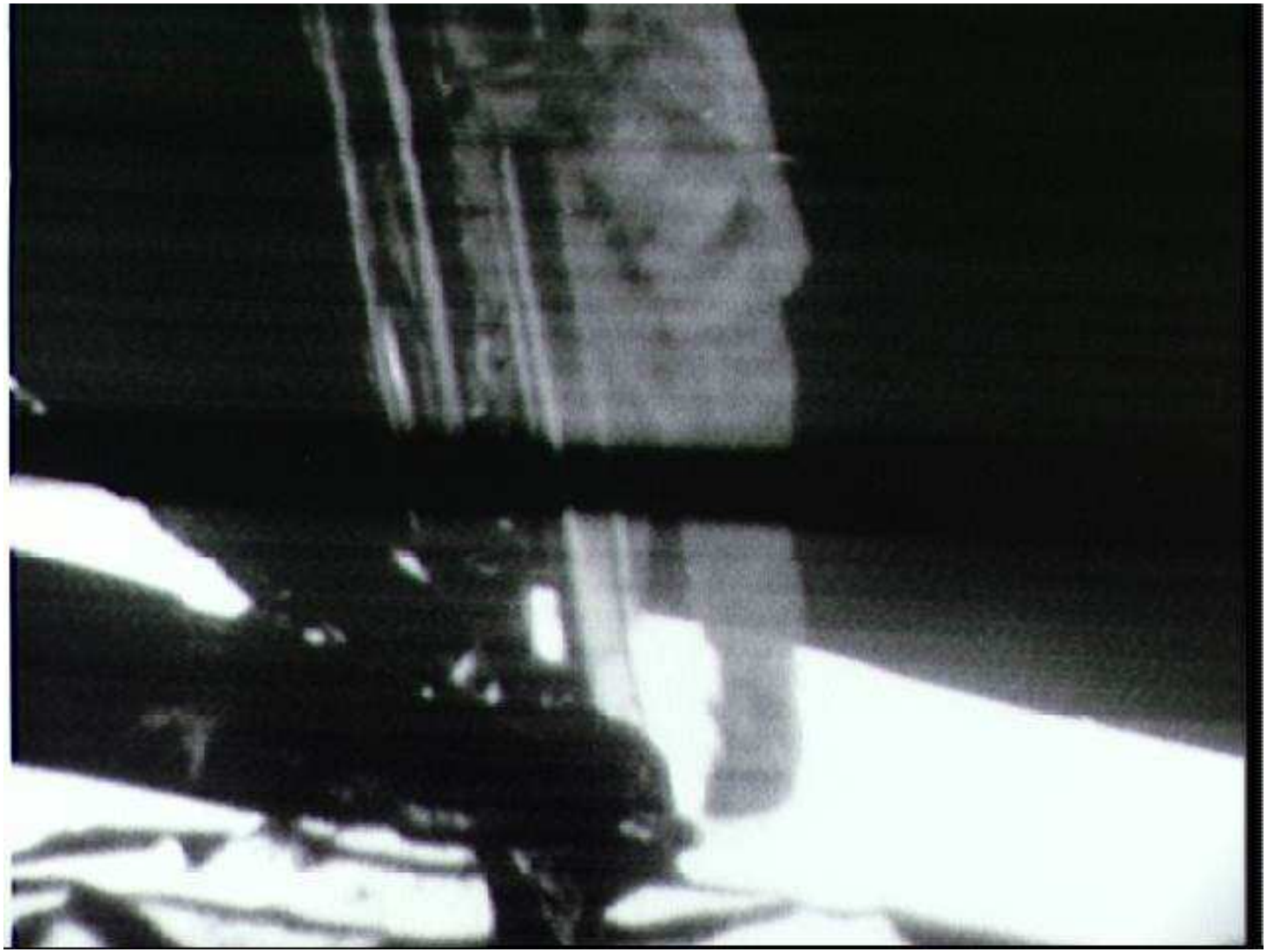
## Corrected First Words

*That's one small step for a man*
*One giant leap for mankind.*

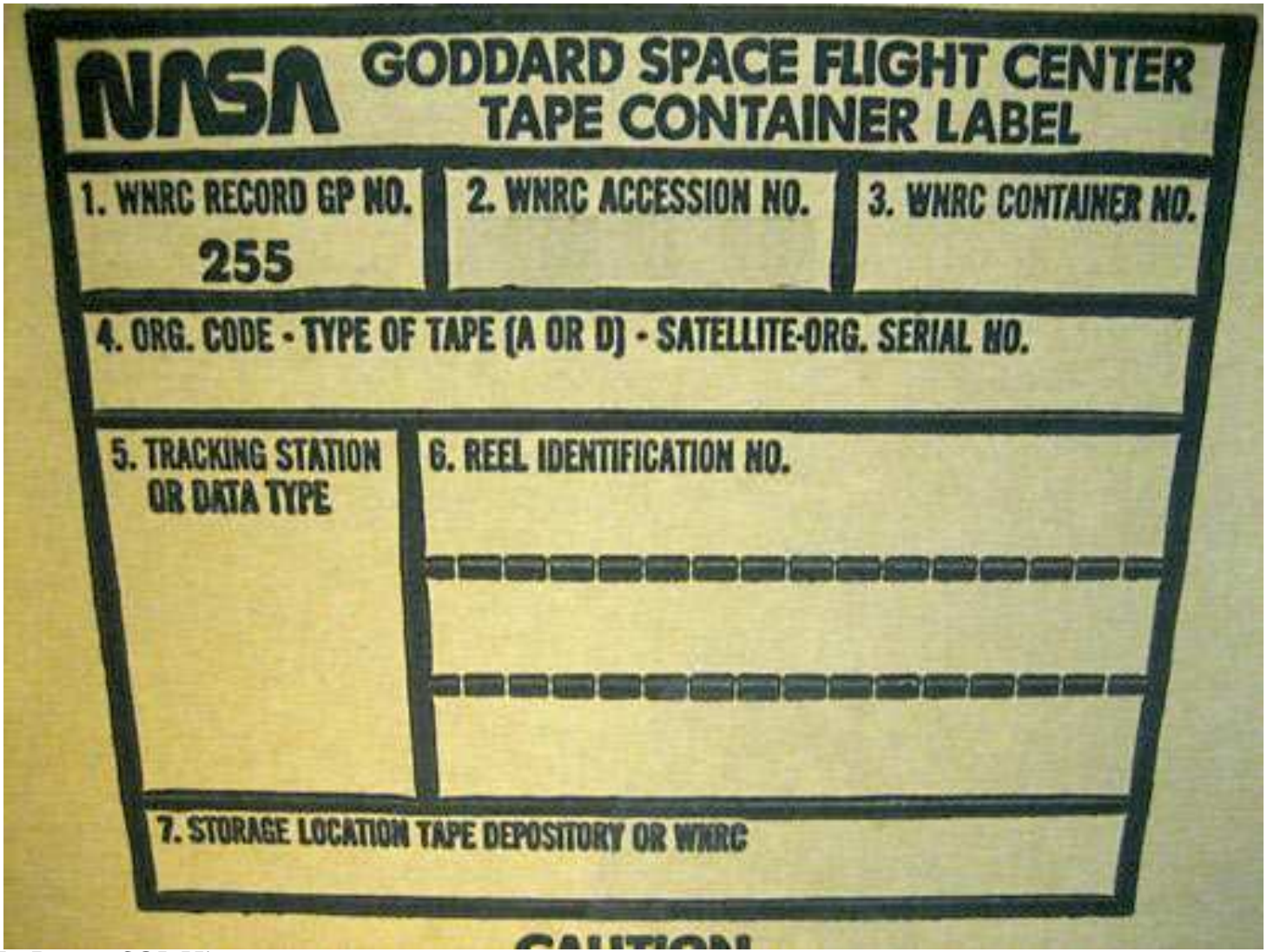
# Moon Camera

# 14-inch Tapes, 10Hz, Slow Scan

# First Steps, 60Hz, Ghostly



# First Steps


NASA
GODDARD SPACE FLIGHT CENTER
TAPE CONTAINER LABEL
1. WNRC RECORD GP NO.
255
2. WNRC ACCESSION NO.
3. WNRC CONTAINER NO.
4. ORG. CODE - TYPE OF TAPE (A OR D) - SATELLITE-ORG. SERIAL NO.
5. TRACKING STATION OR DATA TYPE
6. REEL IDENTIFICATION NO.
7. STORAGE LOCATION TAPE DEPOSITORY OR WNRC
CAUTION

# Finding The Tapes

**1969** Apollow landing tapes recorded

**1970** Tapes moved to Goddard

**2002** Old practice tape found, worked

**2003** Picture of slowscan image found

**2006** Slowscan tapes still not found

# Goddard Space Flight Center

One Giant Screwup for Mankind
*Wired Magazine*

# *Twenty-seven Years Later ... in1996*

# Crash

```
date: 1996/07/19 06:27:59; author: scrappy; state: Exp; lines: +4 -4
Fixes:
'select distinct on' causes backend to crash
submitted by: Chris Dunlop chris@onthe.net.au
```

# Cleanup

date: 1996/11/10 02:59:49; author: momjian; state: Exp; lines: +2 -2
All external **function definitions now have prototypes** that are checked.

date: 1996/10/31 10:12:11; author: scrappy; state: Exp; lines: +3 -
**add #include "postgres.h",** as required by all .c files

date: 1997/09/08 21:42:52; author: momjian; state: Exp; lines: +8 -8
Used modified version of indent that understands **over 100 typedefs.**

date: 1997/09/08 02:22:23; author: momjian; state: Exp; lines: +92 -92
**Another PGINDENT run** that changes variable indenting and case label indenting. Also static variable indenting.

date: 1997/09/07 04:41:09; author: momjian; state: Exp; lines: +357 -336
**Massive commit to run PGINDENT on all *.c and *.h files.**

date: 1997/08/19 21:30:51; author: momjian; state: Exp; lines: +7 -3
**Make functions static** where possible, enclose unused functions in **#ifdef NOT_USED.**

# Monster Macro

```
date: 1997/08/26 23:31:37; author: momjian; state: Exp; lines: +1 -2
Inlined heap_getattr().
```

Breaks AIX, Sun, and SCO compiles.

```
I will admit this is an ugly macro, but this also re-
duced my test times by 75% by reducing the number of function calls.

The compiler I'm using breaks while compiling aclchk.c, in particu-
lar, when  processing the heap_getattr macro.  Yes, I know that it may com-
pile with gcc  (may, be-
cause I have not tried it yet), but I would rather not have to rely on  gcc.  Would it be po
sible to include the non-macro version as a static  function in the appropi-
ate header file?  This would allow the compiler to  in-line the func-
tion if it would provide a speed increase.
```

# More Macro Excitement

```
date: 1997/09/18 20:20:29; author: momjian; state: Exp; lines: +2 -2
Inline memset() as MemSet().
```

# Bye, Bye, Time Travel

```
date: 1997/11/20 23:21:24; author: momjian; state: Exp; lines: +1 -3
Remove all time travel stuff. Small parser cleanup.
```

# NOT NULL Addition Known

```
date: 1997/08/19 04:43:45; author: vadim; state: Exp; lines: +32 -1
NOT NULL implementation (submitted by Robson Paniago de Miranda).
```

# Blockbuster Features

date: 1996/11/13 20:48:28; author: scrappy; state: Exp; lines: +4 -3
Commit of a *MAJOR* patch from Dan McGuirk <djm@indirect.com>
Changes:
* **Unique index capability** works using the syntax 'create unique index'.
* **Duplicate OID's** in the system tables are **removed.** I put little scripts called **'duplicate_oids'** and 'find_oid' in include/catalog that help to find and remove duplicate OID's. I also moved 'unused_oids' from backend/catalog to include/catalog, since it has to be in the same directory as the include files in order to work.
* You can **'SELECT NULL'** to your heart's content.

# Flexibility

```
date: 1997/12/22 05:41:53; author: momjian; state: Exp; lines: +13 -0
Fix for select 1=1 or 2=2, select 1=1 and 2=2, and select sum(2+2).
```

# 64-bit Goodness

date: 1997/03/12 20:58:26; author: scrappy; state: Exp; lines: +2 -2
From: Dan McGuirk <mcguirk@indirect.com>
Subject: [HACKERS] linux/alpha patches

These patches lay the groundwork for a Linux/Alpha port. The **port doesn't actually work unless** you tweak the linker to put **all the pointers in the first 32 bits** of the address space, but it's at least a start. It implements the test-and-set instruction in Alpha assembly, and also fixes a lot of pointer-to-integer conversions, which is probably good anyway.

# Joining On NULLs Crashes

```
date: 1996/08/19 01:07:43; author: scrappy; state: Exp; lines: +11 -1
|This patch fixes a backend crash that happens sometimes when you try to
|join on a field that contains NULL in some rows. Postgres tries to
|compute a hash value of the field you're joining on, but when the field
|is NULL, the pointer it thinks is pointing to the data is really just
|pointing to random memory. This forces the hash value of NULL to be 0.
|
|It seems that nothing matches NULL on joins, even other NULL's (with or
|without this patch). Is that what's supposed to happen?
|
Submitted by: Dan McGuirk <mcguirk@indirect.com>
```

# The Name Game

From: bryanh@giraffe.netgate.net (Bryan Henderson)
Date: Tue, 17 Sep 1996 22:41:09 -0700
Subject: Re: [PG95-DEV] postgres2.0

Here are my thoughts on the name:
- - I want something **pronounceable**, that could actually be used in conversation. FreeRDBMS, as an example, fails that test because it's more of a sentence than a name. I'm not opposed to things with slashes, where you can ignore the junk after the slash if you want.
- - To a lesser extent, I like names that suggest what it is. Like that include "DB" or "SQL". "gres" works pretty well toward that end too.
- - I'm not fond of **"2000"** names. Besides being really cliche now, I find something anticlimactic about naming anything after the last year of a millenium.
- - I'd rather not tack anything onto "Postgres." The problem is that, due to the pronounceability issue above, it will get abbreviated to Postgres, and that **causes confusion** with the original Postgres.
Besides, people familiar with the old Postgres may not even realize that **ours talks SQL**. I know when I went searching for a DBMS, that misled me

for a while.

Based on all that, and other people's input, my favorite among what I've seen so far is **"TiGres"**. However, I would not be disappointed if it were something as boring as "Postgres Version 6."

# Pronouncability

From: "Marc G. Fournier" <scrappy@ki.net>
Date: Wed, 18 Sep 1996 02:03:40 -0400 (EDT)
Subject: Re: [PG95-DEV] postgres2.0

On Tue, 17 Sep 1996, Bryan Henderson wrote:
> Based on all that, and other people's input, my favorite among what I've
> seen so far is **"TiGres"**. However, I would not be disappointed if it were
> something as boring as "Postgres Version 6."
>
Any significance to TiGres, or just sounds neat? :)

How about PostgresV6?

Please...though...just not **Postgres/SQL**...it just **doesn't slide off the tongue nicely :(**

# Anything But Postgres/SQL

From: "Vadim B. Mikheev" <vadim@sable.krasnoyarsk.su>
Date: Wed, 18 Sep 1996 15:04:22 +0800
Subject: Re: [PG95-DEV] postgres2.0

Marc G. Fournier wrote:

...
> How about PostgresV6?

And it's good name. But Ingres & Postgres are different in language (not only),
and Postgres & Postgres'95 too, so why not to change prefix now again ?

**> Please...though...just not Postgres/SQL...it just doesn't slide**

**Vote against too.**

# The Join

From: Gerardo Garcia Cabrero <ggarcia@six.chorro.com.mx>
Date: Thu, 26 Sep 1996 08:07:10 -0600 (CST)
Subject: Re: [PG95]: Re: [PG95-DEV] postgres2.0 name

On Wed, 25 Sep 1996, Drew Einhorn wrote:
> Marc G. Fournier wrote:
> >
> > Since the whole debate has quieted down to nothing, unless there
> > are any major disagreements on just sticking with plain Postgres, after
> > I get the machines moved this weekend, I'd like to work on changing the
> > docs to just drop the 95 part and leave it as being Postgres...
> >
> > ...to me, adding in SQL to it is redundant...
> >
> > Again...if anyone has any disagreements with this, please speak
> > up...
> >

>
> I know I've taken documents containing lists of database systems,
> with brief descriptions. Done a "grep -i sql" and not hit postgres
> even though it was on the list.
>
> Now if you add SQL to the name of postgres, you have a better chance of
> getting a hit as long as the person maintaining the list gets the name
> right.
> Many of the people maintaining these lists try to get the names right,
> but
> don't have time to take a look at the features of all the systems.
>

**postGresql** ?

# A Done Deal

From: "Marc G. Fournier" <scrappy@ki.net>
Date: Tue, 22 Oct 1996 04:35:34 -0400 (EDT)
Subject: Re: [PG95-DEV] postgres.org

On Tue, 22 Oct 1996, Julian Assange wrote:
>
> Does anyone know what is happening with the delegation of the
> postgres domains? Indirect.com was going to hand them over,
> but as yet has not.

Last week, I sent an email off to the 'admin contact' asking
for a status, but have heard nothing...
Which is okay.
We had that major discussion on name changes, and it seemed
that **the majority were in favor of PostgreSQL**...so let's do it.

I'm going to submit a DNS request for PostgreSQL.ORG tonight,

and will work on changing the repositories so that they reflect the
new name, which will get ball rolling.
We basically got it down to two names, out of which I think that
one is the only one that really translates well to a domain, as something
like PostgresV6 would need a new domain each time we started a release
major number :)

Since it is *technically* Postgres v6.0 that we are working on,
releases will be named postgresql-v6.x...
And, as everyone argued...it retains our history, as well as
advertises us as SQL based...

# New Domain Name

From: "Marc G. Fournier" <scrappy@ki.net>
Date: Tue, 22 Oct 1996 05:11:44 -0400 (EDT)
Subject: [PG95-DEV] PostgreSQL.ORG

Hey...
Well, I'm now recovering from a heart attack...

I sent out the **registration request** for PostgreSQL.ORG
no more then 15 minutes ago...I got confirmation that it is complete
5 minutes ago...

DNS is setup, so that should take 12-24 hours to propogate
out...will setup all the virtual WWW/ftp/mail stuff when I'm at
the office tomorrow.

# Redirect

From: "Marc G. Fournier" <scrappy@ki.net>
Date: Thu, 24 Oct 1996 01:37:02 -0400 (EDT)
Subject: [PG95-DEV] {www,ftp}.PostgreSQL.ORG

Hi...
I've just **switched everything over**, and believe it all to work,
hopefully, transparently.

I've created a Redirect from www.ki.net/postgres95 to
www.PostgreSQL.ORG, so that its pretty transparent.
For those mirroring the WWW site (well, only one so far) and
the ftp site, you can now connect anonymously to:
ftp.postgresql.org:/{pub,www}

Hopefully that will make it easy for WWW mirrors, as the one
that Jason setup had to have an account setup for it, which for one or
two is okay, but more then that...

# Optimizer Magic

```
* From: Bruce Momjian <maillist@candle.pha.pa.us>
* To: tgl@sss.pgh.pa.us (Tom Lane)
* Subject: Re: [HACKERS] Optimizer speed and GEQO (was: nested loops in joins)
* Date: Tue, 2 Feb 1999 15:39:51 -0500 (EST)

OK, I have modified the optimizer to count not only the table, but also
the indexes. Patch is applied. The code is:

{
  List *temp;
  int paths_to_consider = 0;
  foreach(temp, outer_rels)
  {
    RelOptInfo *rel = (RelOptInfo *) lfirst(temp);
    paths_to_consider += length(rel->pathlist);
  }
  if (( use_geqo_ ) && paths_to_consider >= _use_geqo_rels_)
```

```
  /* returns _one_ RelOptInfo, so lcons it */
  return lcons(geqo(root), NIL);
}
```

**It is my understanding that it is the size of the pathlist that determines the complexity of the optimization. (Wow, I actually understood that sentence.)**

Tom, can you tell me if this looks good, and also give a suggestion for a possible default value for GEQO optimization. My guess is that 6 now is too low. Sounds like you have a good testbed to do this.
Old posting attached. I will look at the functions you mentioned for possible improvement.

> I have been looking at optimizer runtime using Charles Hornberger's
> example case, and what I find is that the number of tables involved in
> the query is a very inadequate predictor of the optimizer's runtime.
> Thus, it's not surprising that we are getting poor results from using
> number of tables as the GEQO threshold measure.

...

> A fairly decent approximation is that the runtime varies as **the**

> **square** of the number of create_nestloop_path calls. That number
> in turn seems to vary as the **factorial** of the number of tables,
> with a weaker but still strong term involving the number of indexes.
> I understand the square and the factorial terms, but the form of
> the dependency on the index count isn't real clear.

...

> The main thing that jumps out from the profiles is that on these complex
> searches, the optimizer spends practically *all* of its time down inside
> better_path, scanning the list of already known access paths to see if a
> proposed new path is a duplicate of one already known. (Most of the time
> it is not, as shown by the small number of times path_is_cheaper gets
> called from better_path.) This scanning is O(N^2) in the number of paths
> considered, whereas it seems that all the other work is no worse than O(N)
> in the number of paths. The bottom of the duplicate check is equal(),
> which as you can see is getting called a horrendous number of times.
>
> It would be worth our while to try to eliminate this mostly-unsuccessful
> comparison operation.
>
> I wonder whether it would work to simply always add the new path to the
> list, and rely on the later "prune" step to get rid of duplicates?
> The prune code is evidently far more efficient at getting rid of useless

> entries than better_path is, because it's nowhere near the top of the
> profile even though it's processing nearly as many paths.
>
**> Anybody here know much about how the optimizer works? I'm hesitant to**
**> hack on it myself.**

# Optimizer Bug

* From: Tom Lane <tgl@sss.pgh.pa.us>
* To: "Thomas G. Lockhart" <lockhart@alumni.caltech.edu>
* Subject: Re: [HACKERS] Optimizer speed and GEQO (was: nested loops in joins)
* Date: Sat, 06 Feb 1999 12:06:54 -0500

>> Why are we maintaining this huge Path List for every RelOptInfo
>> structure if we only need the cheapest?

I think Bruce is onto something here... keep only the best-so-far
not everything ever generated. That'd get rid of the O(N^2)
comparison behavior.

The only thing I can think of that we might possibly *need* the
whole list for is if it is used as a recursion stopper.
("Oh, I already investigated that alternative, no need to go down
that path again.") It did not look to me like the optimizer does
such a thing, but **I don't claim to understand the code.**

It seems to me that the search space of possible paths is
well-structured and can be enumerated without generation of duplicates.
You've got a known set of tables involved in a query, a fixed set of
possible access methods for each table, and only so many ways to
combine them. So the real question here is why does the optimizer
even need to check for duplicates --- should it not never generate
any to begin with? The profiles I ran a few days ago show that indeed
most of the generated paths are not duplicates, but a small fraction
are duplicates. Why is that? I'd feel a lot closer to understanding
what's going on if we could explain where the duplicates come from.

"Thomas G. Lockhart" <lockhart@alumni.caltech.edu> writes:
> Perhaps having the entire list of plans available makes it easier to
> debug, especially when the stuff was in lisp (since in that environment
> it is easy to traverse and manipulate these lists interactively)...

The optimizer's Lisp heritage is pretty obvious, and in that culture
building a list of everything you're interested in is just The Way To
Do Things. **I doubt anyone realized that keeping the whole list would
turn out to be a performance bottleneck.**

# Optimizer Guess

```
* From: Bruce Momjian <maillist@candle.pha.pa.us>
* To: tgl@sss.pgh.pa.us (Tom Lane)
* Subject: Re: [HACKERS] Optimizer speed and GEQO (was: nested loops in joins)
* Date: Sat, 6 Feb 1999 12:51:52 -0500 (EST)

> >> Why are we maintaining this huge Path List for every RelOptInfo
> >> structure if we only need the cheapest?
>
> I think Bruce is onto something here... keep only the best-so-far
> not everything ever generated. That'd get rid of the O(N^2)
> comparison behavior.
>
> The only thing I can think of that we might possibly *need* the
> whole list for is if it is used as a recursion stopper.
> ("Oh, I already investigated that alternative, no need to go down
> that path again.") It did not look to me like the optimizer does
> such a thing, but I don't claim to understand the code.
```

>
> It seems to me that the search space of possible paths is
> well-structured and can be enumerated without generation of duplicates.
> You've got a known set of tables involved in a query, a fixed set of
> possible access methods for each table, and only so many ways to
> combine them. So the real question here is why does the optimizer
> even need to check for duplicates --- should it not never generate
> any to begin with? The profiles I ran a few days ago show that indeed
> most of the generated paths are not duplicates, but a small fraction
> are duplicates. Why is that? I'd feel a lot closer to understanding
> what's going on if we could explain where the duplicates come from.

**Here is my guess**, and I think it is valid. There are cases where a join of two tables may be more expensive than another plan, but the order of the result may be cheaper to use for a later join than the cheaper plan. That is why I suspect they are doing in better_path():

```
if (samekeys(path->keys, new_path->keys) &&
    equal_path_ordering(&path->p_ordering,
    &new_path->p_ordering))
{
   old_path = path;
```

```
    break;
}
```

So I don't think we can grab the cheapest path right away, but I think the system is keeping around non-cheapest paths with the same result ordering. I also would like to throw away plans that are so expensive that sorting another plan to the desired ordering would be cheaper than using the plan.

**I am only guessing on this.** How often does this 'keep non-cheapest path around' really help?

# Tough Problems

* From: Bruce Momjian <maillist@candle.pha.pa.us>
* Subject: Re: temp tables
* Date: Wed, 10 Dec 1997 22:18:50 -0500 (EST)

I was just reading "Unix Internals: The New Frontiers", and the kernel issues they discuss are very similar to our backend issues.
The reason we have so much trouble is because this stuff is hard.
Jolly, the person who maintained this before Marc brought us together, said this project needs **a few people with lots of time**, not many people with a little time.
I see what he means. **Vadim has helped us keep going by fielding all the tough problems.**

# Temp Tables - Lack of Confidence

* From: Bruce Momjian <maillist@candle.pha.pa.us>
* To: hackers@postgreSQL.org (PostgreSQL-development)
* Subject: TEMP table code
* Date: Thu, 28 Jan 1999 16:38:47 -0500 (EST)

I am attaching a file containing changes needed for adding temp tables to the backend code. This is not a complete patch because I am adding new files and stuff. It is attached just for people to review.

The **basic question is whether this is the proper way** to do temp tables. This implementation never adds the table to the system tables like pg_class and pg_attribute. It does all temp table work by mapping the user table names to system-generated unique table names using the cache lookup code. Fortunately because of the mega-patch from August, almost all system table access is done through the cache. Of course, a table scan of pg_class will not show the temp table because it is not really in pg_class, just in the cache, but there does not seem to be many cases

where this is a bad thing.

I still need to run some more tests and add a few more features, but you get the idea. I hope to apply the patch tomorrow or Saturday.
The only other way I can think of doing temp tables is to actually insert into the system tables, and have some flag that makes those rows only visible to the single backend that created it. We would also have to add a new pg_class column that contained the temp name, and modify pg_class so it could have duplicate table names as long as the temp name was unique. This seemed very unmodular, and would add more complexity to the heap tuple tuple visibility code.
Here is a sample of what it does:

# Wow, It Actually Worked

```
#$ sql test
Welcome to the POSTGRESQL interactive sql monitor:
test=> select * from test;
ERROR: test: Table does not exist.
test=> create temp table test (x int);
CREATE
test=> insert into test values (3);
INSERT 19745 1
test=> \q
#$ sql test
Welcome to the POSTGRESQL interactive sql monitor:
test=> select * from test;
ERROR: test: Table does not exist.
test=>
```

In this example, I create a non-temp table, then mask that with a temp table, then destroy them both:

```
#$ sql test
Welcome to the POSTGRESQL interactive sql monitor:
```

```
test=> create table test (x int);
CREATE
test=> create temp table test (x int);
CREATE
test=> create temp table test (x int);
ERROR: test relation already exists
test=> drop table test;
DROP
test=> drop table test;
DROP
test=>
```

# You Can't Do Temporary Tables Like That

* From: Bruce Momjian <maillist@candle.pha.pa.us>
* To: hackers@postgreSQL.org (PostgreSQL-development)
* Subject: TEMP tables applied
* Date: Mon, 1 Feb 1999 22:57:04 -0500 (EST)

I have applied TEMP tables to the tree, with documenation changes and a regression test.

I am sure there are going to be some features that break with temp tables.

First, there is no pg_type tuple to match the temp table name. There is one to match the system-generated temp table name, pg_temp.pid##.seq##. Temp table do not show up in \d, but you can see them in \dS as pg_temp*.

Not sure if sequences/SERIAL/PRIMARY will work with temp tables. We

either need them to work, or prevent them from being created. Testing will be necessary.

One item I was not 100% happy with. In backend/catalog/heap.c and index.c, there is code that say:

```
/* invalidate cache so non-temp table is masked by temp */
if (istemp)
{
  Oid relid = RelnameFindRelid(relname);
  if (relid != InvalidOid)
  {
    /*
    * This is heavy-handed, but appears necessary bjm 1999/02/01
    * SystemCacheRelationFlushed(relid) is not enough either.
    */
    RelationForgetRelation(relid);
    ResetSystemCache();
  }
}
```

I found that in my regression test where I create a table, and index,

then a temp table and index, when I go to delete the temp index and table, the non-temps are deleted unless I use ResetSystemCache(). If I SystemCacheRelationFlushed(relid), the non-temp table is never deleted from the cache. It only seems to be a problem with indexes.

**Does anyone know why?** Currently, it works, but I am not happy with it. Does anyone understand why the cache would not flush? Can someone debug a call to SystemCacheRelationFlushed(relid) and see why the invalidation is not happening on the non-temp relid.

# The Dreaded 8K Row Limit

```
* From: Christian Rudow <Christian.Rudow@thinx.ch>
* To: John Huttley <john@mwk.co.nz>, PostgreSQL General <pgsql-general@hub.org>
* Subject: Re: [GENERAL] Limitation
* Date: Thu, 24 Jun 1999 07:49:11 +0200

John Huttley wrote:
> PG cannot handle this. see below.
>
> create view product as
> Select code As Stock_Code,
> Substr(Code,1,1) As Process_ID,
> Substr(Code,2,2) As SubProcess_ID,
> Substr(Code,4,1) As SubStrate_ID,
> Substr(Code,5,2) As Length_ID,
> Substr(Code,7,2) As Width_ID,
> Substr(Code,9,2) As Thickness_ID,
> Substr(Code,11,3) As Face_ID,
```

> Substr(Code,14,1) As Facefinish_ID,
> Substr(Code,15,3) As Back_ID
> -- Substr(Code,18,1) As Backfinish_ID
> >From INMASTER;
**> ERROR: DefineQueryRewrite: rule plan string too big.**

John,
I generate a lot of sql create scripts from perl classes.
These scripts tend to hav a lot of whitespaces that make them beautiful to read.

But just for me ... that's what PG means to this :
PQsendQuery() -- query is too long. Maximum length is 8191
So far, I got along very well **just stripping unneeded whitespaces** out of the scripts before i run them.
Probably a simple s/^ // would already help in your case.

# Invisible lztext

* From: Tom Lane <tgl@sss.pgh.pa.us>
* To: Don Baccus <dhogaza@pacifier.com>
* Subject: Re: [HACKERS] interesting observatation regarding views and V7.0
* Date: Wed, 23 Feb 2000 18:46:44 -0500

Don Baccus <dhogaza@pacifier.com> writes:
>> Something else we might consider as a stopgap is to resurrect the
>> **"compressed text" datatype** that Jan wrote, and then removed in
>> anticipation of having TOAST.
> Also...interbase's "text" type is apparently compressed, and that's
> an interesting idea for "text" itself (as opposed to "varchar()" of
> a given size). Someone who just says "text" probably wants to be
> able to stuff as much text into the column as possible, I know
> I do!

**Just quietly make text compressed-under-the-hood, you mean?** Hmm.
Interesting idea, all right, and it wouldn't create any long-term

compatibility problem since users couldn't see it directly. I think
we might have some places in the system that assume char/varchar/text
all have the same internal representation, but that could probably
be fixed without too much grief.
> The price of compression/decompression is to some extent
> balanced by not having to drag as many bytes around during joins
> and sorts and the like.

Also, there could be a threshold: don't bother trying to compress
fields that are less than, say, 1K bytes.
Jan, what do you think? I might be able to find some time to try this,
if you approve of the idea but just don't have cycles to spare.

# Fsync To Disk

* From: "Christopher Kings-Lynne" <chriskl@familyhealth.com.au>
* To: "Hackers List" <pgsql-hackers@postgresql.org>
* Subject: RE: 8192 BLCKSZ ?
* Date: Tue, 28 Nov 2000 09:14:15 +0800

I don't believe it's a performance issue, I believe it's that writes to **blocks greater than 8k cannot be guaranteed 'atomic'** by the operating system. Hence, 32k blocks would break the transactions system. (Or something like that - am I correct?)

# Disk Sector Size

* From: Nathan Myers <ncm@zembu.com>
* To: pgsql-hackers@postgresql.org
* Subject: Re: 8192 BLCKSZ ?
* Date: Mon, 27 Nov 2000 17:49:46 -0800

**Nothing is guaranteed for anything larger than 512 bytes,** and even then you have maybe 1e-13 likelihood of a badly-written block written during a power outage going unnoticed. (That is why the FAQ recommends you invest in a UPS.) If PG crashes, you're covered, regardless of block size. If the OS crashes, you're not. If the power goes out, you're not.

The block size affects how much is written when you change only a single record within a block. When you update a two-byte field in a 100-byte record, do you want to write 32k? (The answer is "maybe".)

# Filesystem Block Size

* From: Tom Lane <tgl@sss.pgh.pa.us>
* To: "Christopher Kings-Lynne" <chriskl@familyhealth.com.au>
* Subject: Re: 8192 BLCKSZ ?
* Date: Tue, 28 Nov 2000 00:38:37 -0500

As Nathan remarks nearby, **it's hard to tell how big a write can be assumed atomic,** unless you have considerable knowledge of your **OS and hardware.** However, on traditional **Unix filesystems (BSD-derived)** it's a pretty certain bet that writes larger than 8K will *not* be atomic, since 8K is the filesystem block size. You don't even need any crash scenario to see why not: just consider running your disk down to zero free space. If there's one block left when you try to add a multi-block page to your table, you are left with a corrupted page, not an unwritten page.

Not sure about the wild-and-wooly world of Linux filesystems... anybody know what the allocation unit is on the popular Linux FSes?

My feeling is that 8K is an entirely reasonable size now that we have TOAST, and so there's no longer much interest in changing the default value of BLCKSZ.

In theory, I think, WAL should reduce the importance of page writes being atomic --- but it still seems like a good idea to ensure that they are as atomic as we can make them.

# The Disk, The Disk

* From: Nathan Myers <ncm@zembu.com>
* To: pgsql-hackers@postgresql.org
* Subject: Re: 8192 BLCKSZ ?
* Date: Tue, 28 Nov 2000 13:01:34 -0800

Not to harp on the subject, but even if you _do_ know a great deal about your OS and hardware, you _still_ can't assume any write is atomic.

To give an idea of what is involved, consider that **modern disk drives routinely re-order writes,** by themselves. You think you have asked for a sequential write of **8K bytes, or 16 sectors,** but the disk might write the first and last sectors first, and then the middle sectors in random order. A block of all zeroes might not be written at all, but just noted in the track metadata. Most disks have a "feature" that they report the write complete as soon as it is in the RAM cache, rather than after the sectors

are on the disk. (It's a "feature" because it makes their benchmarks come out better.) It can usually be turned off, but different vendors have different ways to do it. Have you turned it off on your production drives?

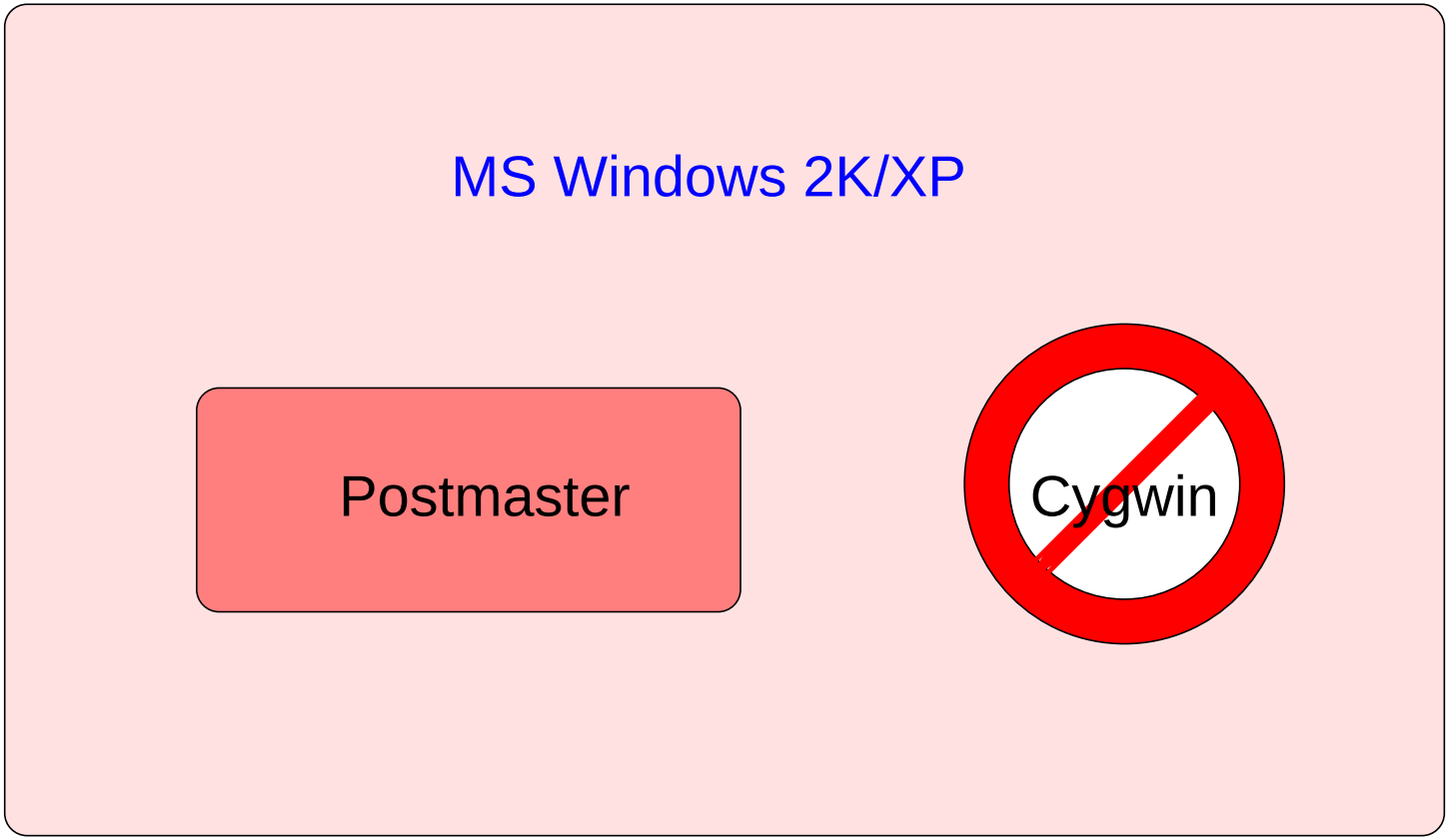
**In the event of a power outage, the drive will stop writing in mid-sector.** If you're lucky, that sector would have a bad checksum if you tried to read it. If the half-written sector happens to contain track metadata, you might have a bigger problem.

---

**The short summary is: for power outage or OS-crash recovery purposes, there is no such thing as atomicity.**

...

# No Cygwin



# Win32 Roadmap

* From: Bruce Momjian <pgman@candle.pha.pa.us>
* To: PostgreSQL-development <pgsql-hackers@postgreSQL.org>
* Subject: Roadmap for a Win32 port
* Date: Wed, 5 Jun 2002 00:33:44 -0400 (EDT)

OK, I think I am now caught up on the Win32/cygwin discussion, and would like to make some remarks.

**First, are we doing enough to support the Win32 platform?** I think the answer is clearly **"no". There are 3-5 groups/companies working on Win32 ports** of PostgreSQL. We always said there would not be PostgreSQL forks if we were doing our job to meet user needs. Well, obviously, a number of groups see a need for a better Win32 port and we aren't meeting that need, so they are. I believe this is one of the few cases where groups are going out on their own because we are falling behind.

So, there is no question in my mind we need to **do more to encourage**

**Win32 ports.** Now, on to the details.

INSTALLER
---
We clearly need an installer that is zero-hassle for users. We need to decide on a direction for this.

GUI
---
We need a slick GUI. pgadmin2 seems to be everyone's favorite, with pgaccess on Win32 also an option. What else do we need here?

BINARY
---
This is the big daddy. It is broken down into several sections:
FORK()
How do we handle fork()? Do we use the cygwin method that copies the whole data segment, or put the global data in shared memory and copy that small part manually after we create a new process?

THREADING

---
Related to fork(), do we implement an optionally threaded postmaster, which eliminates CreateProcess() entirely? I don't think we will have superior performance on Win32 without it. (This would greatly help Solaris as well.)
IPC
We can use Cygwin, MinGW, Apache, or our own code for this. Are there other options?

ENVIRONMENT
---
Lots of our code requires a unix shell and utilities. Will we continue using cygwin for this?

---

As a roadmap, it would be good to get consensus on as many of these items as possible so people can start working in these areas. We can keep a web page of decisions we have made to help rally developers to the project.

# Update Win32 Roadmap

* From: Bruce Momjian <pgman@candle.pha.pa.us>
* To: PostgreSQL-development <pgsql-hackers@postgreSQL.org>
* Subject: Re: Roadmap for a Win32 port
* Date: Wed, 5 Jun 2002 22:57:00 -0400 (EDT)

Here is a **summary of the responses** to my Win32 roadmap. I hope this
will allow further discussion.

---

INSTALLER
---
Cygwin Setup.exe http://cygwin.com
Nullsoft http://www.nullsoft.com/free/nsis/

GUI
---

pgAdmin2 http://pgadmin.postgresql.org/pgadmin2.php?ContentID=1
pgaccess http://pgaccess.org/
Java admin (to be written)
Dev-C++ admin (to be written) http://sourceforge.net/projects/dev-cpp/

BINARY
---
FORK()
cygwin fork() http://cygwin.com
CreateProcess() and copy global area
THREADING
Posix threads
Gnu pth http://www.gnu.org/software/pth/
ST http://state-threads.sourceforge.net/docs/st.html
(single-session multi-threading possible)
(Posix AIO is possible)
IPC
Cygwin http://cygwin.com
MinGW http://www.mingw.org/
ACE http://www.cs.wustl.edu/~schmidt/ACE.html
APR http://apr.apache.org/
Our own

```
ENVIRONMENT
-----------
Cygwin http://cygwin.com
UnxUtils http://unxutils.sourceforge.net/
Write own initdb

IMPLEMENTATIONS
---------------
PostgreSQLe http://hp.vector.co.jp/authors/VA023283/PostgreSQLe.html
Dbexperts http://www.dbexperts.net/postgresql
Connx http://www.connx.com/
gborg http://gborg.postgresql.org/project/winpackage/projdisplay.php
Interbase http://community.borland.com/article/0,1410,23217,00.html
```

# The Win32 Team

# PostgreSQL Releases

8.2.4 8.2.3 8.2.2 8.2.1 8.2 8.1.9 8.1.8 8.1.7 8.1.6 8.1.5
8.1.4 8.1.3 8.1.2 8.1.1 8.1 8.0.13 8.0.12 8.0.11 8.0.10 8.0.9
8.0.8 8.0.7 8.0.6 8.0.5 8.0.4 8.0.3 8.0.2 8.0.1 8.0 7.4.17
7.4.16 7.4.15 7.4.14 7.4.13 7.4.12 7.4.11 7.4.10 7.4.9 7.4.8
7.4.7 7.4.6 7.4.5 7.4.4 7.4.3 7.4.2 7.4.1 7.4 7.3.19 7.3.18
7.3.17 7.3.16 7.3.15 7.3.14 7.3.13 7.3.12 7.3.11 7.3.10
7.3.9 7.3.8 7.3.7 7.3.6 7.3.5 7.3.4 7.3.3 7.3.2 7.3.1 7.3
7.2.8 7.2.7 7.2.6 7.2.5 7.2.4 7.2.3 7.2.2 7.2.1 7.2 7.1.3
7.1.2 7.1.1 7.1 7.0.3 7.0.2 7.0.1 7.0 6.5.3 6.5.2 6.5.1 6.5
6.4.2 6.4.1 6.4 6.3.2 6.3.1 6.3 6.2.1 6.2 6.1.1 6.1 6.0 1.09 1.02
1.01 1.0 0.03 0.02 0.01

# How We Remember It