Postgres OnLine Journal: March 2010 / April 2010

An in-depth Exploration of the PostgreSQL Open Source Database

Table Of Contents

From the Editors

What is New in PostGIS Land

What's new and upcoming in PostgreSQL

CatchMe - Microsoft SQL Server for Unix and Linux

Basics

In Defense of varchar(x)

Import fixed width data into PostgreSQL with just PSQL *Beginner*

Using PostgreSQL Contribs

PostGIS Raster its on: 10 things you can do NOW with raster

Reader Comments

A Product of Paragon Corporation
http://www.paragoncorporation.com/
http://www.postgresonline.com/

What is New in PostGIS Land

This month we we will be giving two mini-tutorials at PgCon East 2010 on Saturday, March 27th. The topic of the talks will be, you guessed it, *PostGIS*. We have changed our Beyond talk to PostGIS: Adding spatial support to PostgreSQL to a beginner focus instead of an intermediate focus. Topic content will be more or less the same but focused more on people new to spatial database analysis. Our web applications talk will cater more to the web developer trying to integrate PostGIS in their web applications.

Marcus Rouhani of the Federal Aviation Administation will also be talking about the Airport GIS project and migration from Oracle to PostgreSQL.

On a somewhat related note, we also hope to be finished with all the chapters of our upcoming book this month. We just completed the first draft of our Chapter 10: PostgreSQL Add-ons and ancillary tools. After some back and forth with our editor, this will be up on MEAP, available for read and comments for early book buyers. Still two more chapters to finish after that before we get to the polishing of the text, images, layout and final print version.

Our publisher Manning is running a 50% off sale this Friday (tomorrow or is it today) on any MEAP book and they have a lot of interesting ones in the pipeline (including ours).

Waiting for PostGIS 2.0

The OSGEO just completed a recent coding sprint in New York. The New York sprint was a meeting of the minds of OSGEO people from various projects -- PostGIS, Mapserver, Geoserver, OpenLayers, GDAL, and some others were represented. Sadly we were not able to attend this one. A summary of the sprint with a PostGIS bent can be found on Olivier Courtin's New York sprint summary (Original French Version) and Olivier Courtin's New York sprint summary (Google English translation) and Paul's New York sprint summary.

Those in participation on the PostGIS front and diligently working away at new PostGIS 2.0 features were:
Some old faces:

- Olivier Courtin of Oslandia - reorganizing the KML,GML functions and working out the details of polyhedral surfaces suitable for storing 3D buildings and other 3D objects as well as (export features to Collada, X3D etc) needed for CityGML and BIM like efforts. We are particularly excited about polyhedral surface support and generally improved support for 3D in PostGIS 2.0 since we have/had a need for this with one of our current projects. The lack of this feature has meant no compelling reason as of yet to switch this application to a PostgreSQL back-end.
- Paul Ramsey of OpenGeo - working on start of restructure of the way we store geometries and multi-dimensional index support so we can support more types in the core and better support 3D and getting our WKT representation more MM OGC compliant.
- Sandro Santilli AKA strk working remotely by IRC. Mostly working out ST_CleanGeometry robust function for fixing malformed geometries and various other GEOS integrations.

Some new faces

- Jeff Adams of Azavea working on Lat Lon formatter and also improving our CUnit unit testing system.
- David Zwarg also of Azavea working on WKT Raster. Particularly he started work on ST_MapAlgebra.

For those who are interested in knowing what's planned for PostGIS 2.0, you can check out an abbreviated summary PostGIS 2.0 roadmap.

Back to Table Of Contents

CatchMe - Microsoft SQL Server for Unix and Linux

Today Microsoft unveiled their top secret project code named **CatchMe**. This is their new flagship database for Linux and Unix based on predominantly the PostgreSQL 9.0 code base, but with an emulation layer that makes it behave like SQL Server 2008 R2. Unlike the Windows SQL Server 2008 R2 product, this version is completely free and open source under the Microsoft Public License (Ms-PL). Downloads for the RCs of these will be available soon. Please stay tuned.

Reporter Dat A. Base managed to get an exclusive interview with the head of the project, Quasi Modo. The transcript follows:

- **Q: Dat:** *Why is Microsoft now going after the Linux business so late in the game?*
  A: Quasi: We like showing up late to all parties. It gives us an idea what to wear.
- **Q: Dat:** *Why are you making this Free and Open Source - MPL?*
  A: Quasi: We discovered we make more money selling software, support contracts, and consulting for SharePoint Team Services, Microsoft CRM, Microsoft Dynamics and Office than we do selling SQL Server. SQL Server is just a tool for us to upsell our Office (via Excel Pivot integration) and other products. Since many people are migrating to Linux (at least for server), we are losing a large body of potential buyers for our other products to Oracle's half-assed BI products. If we could capture this market share -- we can be the dominant force in the COTS and services arena and outstrip Oracle in their efforts.
- **Q: Dat:** Why did you base your Linux product on PostgreSQL instead of SQL Server 2008?
  A: Quasi: Well its no big secret that SQL Server came from Sybase a predominantly Unix/Linux based server. Sybase itself was developed by Berkeley students working on Ingres and Postgres projects. As such SQL Server and PostgreSQL are more alike than what you would expect. We have all these years been baking SQL Server into our windows ecosystem, such that its next to impossible to get it to work on Linux/Unix without significant rework. PostgreSQL has on the other hand made theirs a cross-platform product, very ANSI SQL compliant similar to SQL Server, and yet works on many OSes. It was easier for us to make PostgreSQL look like SQL Server than to get SQL Server working on Linux/Unix. On top of that PostgreSQL rich pluggable language architecture allows us to run .NET code easily in PostgreSQL. We have also implemented an Open Source PL language called **PL/Redmond** that is syntactically compatible to our Transact-SQL offering. Postgres pluggable authentication system also allows us to keep on using Active Directory for some kinds of authentication and build in other Authentication schemes. In fact we plan to eventually rework our core SQL Server base based on PostgreSQL similar to what EnterpriseDb has done for their Oracle emulation layer. We call this *rediscovering our roots and reinventing ourselves.*
- **Q: Dat:** *How are you going to get SharePoint and Microsoft CRM to work on Linux?*
  A: Quasi: Well they almost do already using Mono.NET. In fact we estimate there is very little we need to do to make them work on Linux.

Back to Table Of Contents CatchMe - Microsoft SQL Server for Unix and Linux Reader Comments

In Defense of varchar(x)

This is a rebuttal to depesz's charx, varcharx, varchar, and text and David Fetter's varchar(n) considered harmful. I respect both depesz and David and in fact enjoy reading their blogs. We just have deferring opinions on the topic.

For starters, I am pretty tired of the following sentiments from some PostgreSQL people:

- 99% of the people who choose varchar(x) over text in PostgreSQL in most cases are just ignorant folk and don't realize that text is just as fast if not faster than varchar in PostgreSQL.
- *stuff your most despised database here* compatibility is not high on my priority list.
- It is unfortunate you have to work with the crappy tools you work with that can't see the beauty in PostgreSQL text implementation. Just get something better that treats PostgreSQL as the superior creature it is.

I don't like these sentiments because, it gives the perception of PostgreSQL users as a bunch of prissy people who expect everyone to step up to their level of misguided sophistication. We shouldn't cater to the lowest common denominator always, but when the lowest common denominator approach is easier to write, easier to read, and achieves the same purpose, why should people be forced to do it any other way. Domains and triggers are great things, but they are overkill in most cases and are harder to inspect.

I think depesz nails it on the head when he points out the main reason why varchar(x) is worse than text in PostgreSQL. It is because if you have a big table, changing the size of a varchar field takes eons in PostgreSQL. This is not so much the case with other databases we have worked with. To me **this is not a problem with everyone else, but a problem with PostgreSQL, and something that should be improved on**. In fact you would think this would be simpler in PostgreSQL than any other database since varchar and text are implemented essentially the same. The main difference it seems is that a text field always results in creation of an accompanying toast table where as varchar doesn't always.

Instead newcomers are patted on the head and told *You young'in, just use text and throw a domain around it if you want to limit size.*

I am not going to write a domain or trigger everytime I have to limit the size of a field, for several reasons and listed in priority.

1. Easily to access and self-enforcing meta data is important. In many cases, its more important than performance.

   When I look at a CREATE TABLE statement or open up a table in a designer, the size there gives a sense of what goes in there. One can query practically any databases system tables (and the ANSI compliant ones have information_schema. columns) and get the size of a field with a single query. This simplicity in portability is a beautiful thing. This simplicity is important for people developing cross database tools. The size gives a sense of what is safe to select and what is not. The size gives auto form generators a sense of the width to make a field and limits to set on what can be put in it. It allows the tool to quickly determine whether to put in a regular text box or a text area. varchar(...) forces everyone to state this exactly the same across all databases. I don't need to figure out how to make sense of the logic embedded in a domain object or trigger or the fact that people can state the same thing differently with domains. If you don't care about the ease with which tool developers can extend their tools to work with your DBMS or making the common place needs easy, then you don't care about growing your community.
2. Using domains when you are talking about 20 fields is way too much typing if all the fields need to be different sizes. varchar(x) has a semantic meaning that is easy to define in 2 characters or less. Domains do not.
3. And to David -- "In the real world things have min sizes as well as max sizes" Yes, but maxlength in most cases is the most important thing. So important that it dwarfs the relevancy of other minutia. Most cases we don't bother with all the other nuances because it clutters the most important thing MAX LENGTH. Less is better than more. I don't want people stuffing the kitch sink in a first name field. Many people like me have t100s of first names they got in a naming ceremony. If they want to call themselves ! or @ or some 4 letter word, I really don't care, but if you are known by a 100 different names, I really only want to know about one or two of them, not all of them :)
4. I am a lazy control freak who loves self-documenting structures. I like order. I like things done consistently and I don't expect to work hard to get that. Being a lazy control freak means I have more time to trick others and machines into doing my work for me so I can lounge around sipping long island iced teas :).

So in short I expect my relational database, not only to provide me good performance, but to provide me structure and ease of use as well. I like my crappy tools. My tools do what they are designed to do and I expect PostgreSQL to do the same without me having to jump thru hoops. I want easy walls I can put up and tear down. I choose my foreign key constraints and my field sizes

carefully -- because they do more than just perform well. They are self-documenting objects that can draw themselves on a screen and are always consistent with the simple rules I set up. If I set a cascade update/delete, it means I expect the child data to go away when the parent does. If I set a foreign key with restrict delete, it means if the child has data, then the parent is way too important to be allowed to be deleted lightly. If I use a varchar(x) -- it means I want a max of x characters in that field. If I use a char(n), it means I expect the data to be exactly (n) characters long - which admittedly is rare.

Back to Table Of Contents In Defense of varchar(x) Reader Comments

Import fixed width data into PostgreSQL with just PSQL *Beginner*

Fixed width data is probably the most annoying data to import because you need some mechanism to break the columns at the column boundaries. A lot of people bring this kind of data into a tool such as OpenOffice, Excel or MS Access, massage it into a delimeted format and then pull it in with PostgreSQL copy command or some other means. There is another way and one that doesn't require anything else aside from what gets packaged with PostgreSQL. We will demonstrate this way.

Its quite simple. Pull each record in as a single column and then spit it into the columns you want with plain old SQL. We'll demonstrate this by importing Census data places fixed width file.

Although this technique we have is focused on PostgreSQL, its pretty easy to do the same steps in any other relational database.

Both David Fetter and Dimitri Fontaine have demonstrated other approaches of doing this as well so check theirs out.

UPDATE

- David Fetter - psql, Paste, Perl: Pefficiency!
- Dimitri Fontaine - Import fixed width data with pgloader

The copy commands

We will be using psql to import our data and the psql client side \copy command. Before we get into the specifics, we want to clarify a very important distinction here, that seems to confuse quite a few people new to PostgreSQL.

There are 2 kinds of copy commands. There is the SQL COPY command built into PostgreSQL server and SQL dialect and there is the client \copy command built into the **psql** client tool. Both can be used to import and export data and take more or less the same arguments. However in some cases you may be able to use one and not the other. Below is a brief summary of how they defer

SQL Copy

- Runs under the server context -- the postgres daemon account
- file path is relative to the server, not the person running it even if you are doing it from psql.
- postgres daemon account needs to have rights to read the file being imported.
- Person calling it needs to have super user rights
- Initiated with COPY .... and pretty much valid anywhere you can run PostgreSQL SQL commands

psql client \copy command

- Runs under the client OS context -- OS account of the person running the command
- file path is relative to the client, the person running the command, so for example if you are using psql in windows and your server is Linux, the path will be like C:/path/to/file and the file has to exist in your local windows workstation, not the server.
- You don't need to be logged in as a postgres super user to use this command, though you need write rights to the database you are importing data to.
- The OS account you are logged in as needs to have rights to read the file being imported. postgres daemon account need not have rights.
- Initiated with \copy and only valid in psql or some psql like variant built with psql plumbing.

More extensive details can be found on the PostgreSQL wiki copy command. The copy commands have been enhanced in newer versions of PostgreSQL, so you may want to check out the official manual for your version as well.

The Data

For our following exercises we'll import US Census 2000 places.zip file which is located at http://www.census.gov/geo/www/gazetteer/places2k.html. This is a fixed width file with record layout defined at http://www.census.gov/geo/www/gazetteer/places2k.

html#places.

Specs for reference look as follows:

```
   * Columns 1-2: United States Postal Service State Abbreviation
    * Columns 3-4: State Federal Information Processing Standard (FIPS) code
    * Columns 5-9: Place FIPS Code
    * Columns 10-73: Name
    * Columns 74-82: Total Population (2000)
    * Columns 83-91: Total Housing Units (2000)
    * Columns 92-105: Land Area (square meters) - Created for statistical purposes only.
    * Columns 106-119: Water Area(square meters) - Created for statistical purposes only.
    * Columns 120-131: Land Area (square miles) - Created for statistical purposes only.
    * Columns 132-143: Water Area (square miles) - Created for statistical purposes only.
    * Columns 144-153: Latitude (decimal degrees) First character is blank or "-" denoting North or South
latitude respectively
    * Columns 154-164: Longitude (decimal degrees) First character is blank or "-" denoting East or West
longitude respectively
```

Here are the steps:

1. Unzip file
2. Launch a psql shell. TIP: *If you are using PgAdmin III, version 1.9 and above has a psql icon located in* ***Plugins*** *that will open up a PSQL shell and connect you to the selected database.*
3. Create table that holds data for staging. Just create a table with a single column that allows unconstrained text.
4. `CREATE TABLE places_staging(data text);`
5. Now use psql to import it in. For this example, we set the client_encoding to latin1 since the file is in latin1 encoding. Your client_encoding should match that of the file, otherwise you may experience errors such as invalid for utf-8 etc. We also specified a bogus column delimiter of '|' since we figure our data has no '|' in it. By default copy would assume tab delimeted columns.

```
set client_encoding = 'latin1';
\copy places_staging FROM C:/censusdata/places2k.txt DELIMITER AS '|'
```

6. Next we create our final table and we don't really care about a lot of the fields so we will just import the fields we care about. Here we use the super useful **substring** function to denote where to start the text and how many characters to pick up.

```
CREATE TABLE places(usps char(2) NOT NULL,
    fips char(2) NOT NULL,
    fips_code char(5),
    loc_name varchar(64));
INSERT INTO places(usps, fips, fips_code, loc_name)
    SELECT substring(data,1,2) As usps, substring(data,3,2) As fips, substring(data,3,5) As
fips_code,
        trim(substring(data, 10,64)) As loc_name
    FROM places_staging;
```

If you did it right, you should get an answer like below which should match the number of lines in the file:

```
INSERT 0 25375
```

Back to Table Of Contents Import fixed width data into PostgreSQL with just PSQL Reader Comments

PostGIS Raster its on: 10 things you can do NOW with raster

We just finished the first draft of the last chapter of our book: First look at PostGIS WKT Raster. This completes our hard-core writing and now on to more drafting, polishing all the chapters. In Chapter 13 we demonstrate how to use PostGIS WKT Raster functions by example and cross breed with PostGIS geometry functionality. I was pleasantly surprised to see how nicely the raster and geometry functions play together.

We had intended this chapter to be short about 20 pages in length, because how much can one say about pixels and pictures. As it turns out, a lot. Rasters are more versatile than their picture portrayal on a screen. Rasters are a class of structured storage suitable for representing any numeric, cell based data where each cell has one or more numeric properties (the bands). This covers quite a bit of data you collect with remote sensing and other electronic instrumentation. We had to stretch to over 30 pages; even then we felt we were missing some critical examples.

There is a lot of useful functionality in PostGIS WKT Raster already and should make a lot of people looking for raster support in PostgreSQL very happy. Although the chapter may portray some scenes of violence and torture inflicted on elephants, you can rest assured that it is pure illusion and no real elephants or blue elephant dolls were harmed in the making of this chapter.

As a side note -- our book is now listed on Amazon PostGIS in Action. It is not available in hard-copy yet,but you can pre-order and of course you can order from PostGIS in Action from Manning directly to get the chapter drafts we have posted, updates as we polish them, and the final book when it comes out in hard print.

The Amazon listing would have been so much more exciting, had they not stripped me of my last name or had Leo married to himself.
UPDATE: It appears I now have a last name again
In hind sight, I suppose OBE is more commonly seen as a title of honor rather than a last name, so its only fitting that I should be stripped of mine and Tim Berners-Lee gets it tacked on at the end of his name.

To find out more about PostGIS WKT Raster, we encourage you to check out these links.

- PostGIS WKT Raster Home Page
- PostGIS WKT Raster Frequently asked questions
- WKT Raster Reference

Now we'll itemize 10 things you can do now with PostGIS WKT Raster. In order to use PostGIS WKT Raster, you need PostGIS 1.3.5 or above. Preferably 1.4 or 1.5 or 2.0 alpha.

PostGIS WKT Raster is currently packaged as a separate library and we have windows binaries available.

10 things you can do now with PostGIS WKT Raster

PostGIS WKT Raster introduces a new PostgreSQL datatype called **raster** which is a companion to PostGIS geometry and geography with its own set of functions for working with raster data and interoperating with geometry like objects.

1. You can load most any kind of raster in your PostgreSQL database with GDAL and the gdal2wktraster.py packaged loader, including whole coverages, chopping bigger rasters into smaller rasters, or creating overview tables as part of the load process.
2. You can also store raster data outside the db and just reference it from inside. Speed doing processing will be slower, but you can share the files.
3. You can export your raster data and portions of it or select bands of it to pretty much any format that GDAL supports. Caveats, some kinds of raster band pixel types just don't port well to some raster formats and the exporter doesn't currently support irregularly blocked rasters (though you can store such things in your database).
4. You can read pixel values easily and you can do pixel sampling for selelct areas to make things go much much faster at loss of some precision, as well as other raster properties like geometric extent, pixel size in geometric coordinate units, raster width/height in pixels .
5. You can georeference the rasters as well as setting some other properties. The ST_SetValue function for setting individual

pixels is still in the works.
6. You can intersect rasters with PostGIS geometries.
7. You can polygonize rasters or portions based on pixel values and other attributes. Right now you have to write some sql or plpgsql to do this, but as we speak - Jorge Arevalo and others in the Raster team are working on a ST_DumpAsPolygons, ST_Polygon, as well as others in that family to streamline this process.
8. The new raster type supports 13 different pixel band types as documented ST_BandPixelType which includes pixel band types that store floating point values.
9. No real limit on the number of bands you can have per raster to my knowledge.
10. You can view regularly blocked rasters in Mapserver by defining a PostGIS WKT Raster layer.

We would like to thank all those who have worked on the PostGIS WKT Raster subproject and who have responded to our endless questions:

Developers

- Pierre Racine - lead architect of the project
- Mateusz Loskot - wrote the gdal2wktraster.py loader and many getter functions. Work sponsored by CadCorp
- Sandro Santilli aka (strk) - wrote the in-memory api, build scripts and PostgreSQL connectors
- Jorge Arevalo - wrote GDAL driver and doing a ton of work now. (Work sponsored by Google Summer of Code and now DEIMOS Space)
- David Zwarg - wrote many setter functions. Work sponsored by Azavea

General monetary or related support Sponsors not already mentioned

- Steve Cumming and University Laval (where Pierre works too)
- Michigan Technology & Research Institute
- OSGEO for hosting etc.
- A very special thanks to Frank Warmerdam who founded the Geospatial Data Abstraction Library (GDAL) project which much of the raster support relies on and also provides FWTools binary builds of GDAL, and Mapserver for windows that has the PostGIS WKT Raster driver already compiled in.
- Of course PostGIS and PostgreSQL for providing a very extensible platform to build on.

Back to Table Of Contents PostGIS Raster its on: 10 things you can do NOW with raster Reader Comments

Reader Comments

CatchMe - Microsoft SQL Server for Unix and Linux

topsy.com

uberVU - social comments
This post was mentioned on Twitter by planetpostgres: Leo Hsu and Regina Obe: CatchMe - Microsoft SQL Server for Unix and Linux http://tr.im/U6R4

Gouken
April Fools!!

Wilson
Hi!!

How about windows for linux?? Is It finished??

RARARA RARARA RARARA

revistalinux.net

In Defense of varchar(x)

Ben DeMott
Very valid points Leo, thanks for taking the time to write this. I came to PostgreSQL from MySQL - The "switch" was daunting enough, it took several months for me to realize how to start doing things the Postgres way. If the features weren't somewhat similar, and the data-types somewhat transferable the switch would have been much, much more difficult.

Jon Erdman (aka StuckMojo)
I will agree that it *should* be faster to alter the length of a varchar, near instant in fact. We should fix that.

However...

I've spent too much time having to futz with field sizes because of changing requirements or bad guesses as to the maximum size, especially when it often doesn't really matter as far as the db is concerned.

Therefore I now much prefer to simply use text and not constrain the length at all in the schema. I instead enforce that at the application layer.

I can already hear the "but, but...", but max length is just not something I worry about. Anyone who would want to enter 100 names into my name field will not be having direct access to my db. Use the nice app I've written, thank you.

Aside from that I am constraint crazy. But max length just changes too much, and seems pointless if you're going to sidestep that by setting it way too high anyway.

Do it at the app layer and be done with it. And screw all that work in making domains and such. In the 4 places it's that important, use a CHECK().

On a side note, I got a nice giggle out of the "tired of the following sentiments..." bit, as it definitely sounds like many PG people I know ;-)

depesz
First of all, a bit offtopic - could you s/hubert/depesz/ig ?

Second - I absolutely agree that it should be fixed in Pg. I actually feel ashamed when I have to explain this to someone - especially someone who knows that the binary format is the same.

As for tools - I don't have this particular itch, as my tool of choice is psql, but I do feel your pain, and I think it's a valid point when considering your datatypes and schema.

And lastly - all my advices, and posts are generally targeted at new users of Pg - experiences dbas, know that there is no rule without exceptions. And I still believe that newbies should stick with text - as long as it will take our fine developers to fix size altering - just to avoid the problem when they will need to make this space for names a little bit longer.

Regina
depesz,
Fixed well for this article anyway :). I don't agree newbies are better with text particularly those new to databases, because it teaches new PostgreSQL users to focus on implementation details of a product. For me, if I'm using PostgreSQL as a tool in

teaching relational databases, which I really hope a lot of university's will do, because it is a great example of a good relational database with sound standard ANSI support, I don't want them to focus on teaching PostgreSQL specific work-arounds for problems. When the fine PostgreSQL developers improve on it, that lesson will be obsolete anyway.

I think its one of those things like you said you have to weigh, and in most cases, its rare I have to increase the size of a field, so it only irks me when that rare moment happens. But I think as you said putting it out there as a caution is good.

Regina
That would be all well and good in some cases, except that in many of our cases, we have the same database feeding 4 or 5 applications in different locations.

Its easier to centralize that in the database for us rather than coding it 5 times. If you have to feed to government systems with their silly regulations of size, its nice to know that your structure already meets the criteria and you don't need to check it when you pass it off.

Robert Treat
I think the real problem stems from a conflict between relational theory and the sql standard.

Your claim is that the max size of a field is the most important characteristic, but I don't believe that is true, it only seems that way. People have long needed to add max boundary limits based on performance / storage concerns, as an implementation detail of the underlying systems.
This means that the tools around max length have become very pervasive and easy to use; so even "crappy tools" and "lowest common denominator systems" have pretty standard meta-data oriented support for max length. This makes it easy to declare this piece of metadata, and so the cycle continues.

But's lets pretend that max size was the most important characteristic. Even then, what is the right size for a first_name field? 10 characters? 20? 100? It's pretty arbitrary really, and setting it to _anything_ makes the application worse (by introducing arbitrary limits). You really do want an unbounded type, but the sql standard doesn't give you varchar() or varchar(max). If you want to add varchar, you are stuck making it "hopefully long enough". Poor data modeling indeed.

So, I'd love to see varchar() adopted by sql standard, and then have all of the tools support it to mean unbounded varchar data; but until that happens varchar is no more correct than text.

Regina
Many of us don't care about being absolutely correct, but being fuzzy correct.
By that I mean if you dragged a 1000 people in a room and asked them what should be the max length of a first name,
after about 150 - no one would be raising their hands. So in short somewhere between 20 and 100 is your optimal fuzzy correct answer.

What is the max length of a tweet. Well it has a maxlength right, its not infinite. The max length seems somewhat arbitrary but the fact it exists
and is under n characters has benefits. It forces people to just state the most critical parts, it prevents information overload, and from a
storage/band width perspective, it means you can easily estimate the size of a page with 1000 tweets.

That brings up the performance benefit of limiting text. I can better predict the bandwidth performance of my queries
which is particularly important for web paging or if you have to transfer data to a third party source.
It also means unsuitably large text is not going to mess up the look of my page by taking up two lines instead of 1.

Josh Berkus
Regina,

The problem is that while max-length is a sometimes-useful restriction on a field, it is not sufficient. For example, you say:

"It also means unsuitably large text is not going to mess up the look of my page by taking up two lines instead of 1."

But, it actually does not mean anything of the sort. If you've set varchar(15), I can easily put 5 linebreak characters ... or 4 ... in there, messing up your formatting. So max-length doesn't really help you *at all* there. You need a regex mask.

What VARCHAR(#) does is give the database designer the *illusion* of control over input without actually giving him/her any real control.

That part of the standard, like numeric primary keys, dates to the storage limitations of the databases of the 1980's, when counting characters was very important since you only had 16MB of disk space. It's quite outdated now.

All that being said, the awkward handling of subscript data types (like VARCHAR(#)) in ALTER TABLE in PostgreSQL is and has been a TODO. And it would be really good to fix it, it just requires a substantial refactoring of some very messy code.

--Josh

Regina
Josh,

Its good to hear that this is on the TODO. Getting back to your point about *illusions*.

First the issue you describe about people stuffing new lines in a size constrained input field rarely happens for 2 reasons. 1) The input fields don't allow new lines unless they are textarea.
2) Even when you html escape text (or at least my crappy special html escape function doesn't), it keeps new line as is which promptly gets ignore by the html standard
unless you PRE them.
As a side benefit -- Those people trying to post their own data with their own forms
will be trapped by the database when they try to stuff in a 1000 character name in a field that only accepts 20 characters. So to me -- its not an illusion, its
an extra security blanket. Its not a perfect blanket, but its a blanket.

On a side note, I can't tell you the countless number of times I have been saved when someone gives a me a data feed and they happened to
innocently switch the order of the first name field and a long unconstrained description text field. The import throws an error and sends me an email. I don't need to put in any extra filters and all that
to get this benefit. I don't have to wonder why suddenly my page is randomly slower because 100 of my records have big chunks of text in the first name field that I had assumed to be short
and safe to display in a list.

I will reiterate, I am not looking for absolute control of input, but what I consider good enough control without lifting too many fingers.
Unconstrained text just doesn't do that for me and it doesn't do it for a lot of people I have talked to. You can argue about limitations
of the 1980s, but some limitations are good. Storage -- particularly bandwidth is still a concern. Its still a concern on my crappy phone. Its still concern for my
limited mind who doesn't want to decipher the meaning of a 1000 character first name field :).
A relational database is a 40 year old limiting technology, but I still choose it in most cases over less limiting
NoSQL databases. I want limits even if you consider they are self-imposed ones.

Matt
I agree, schema should be meaningful.

If you want to use TEXT to capture a field that can never contain anything more than a 4 character alpha code, then likewise you should have no problem using NUMERIC to capture anything that contained nothing but digits. You realize, this becomes a regressive pattern: why not just use text for each field? Even primary keys using BIGSERIALs can be modified to use DEFAULT NEXTVAL('silly_id_seq')::TEXT!

Or you could just create a single field of TEXT and dump your content in as XML or JSON structures...

Ah heck, lets just dump it all in a flat file and write our own data parser and search tools....

No - not for me.

Arguments like "I underestimated the width" are not valid. The speak more to the lack of domain knowledge than to the limitation of the database. Having to put up with waiting for a column resize to complete is a natural consequence of not having done the proper research ahead of time.

You will be a better engineer having learned the lesson.

Regina
I agree with all you are saying. In fact its rare that I have to change the column size and when I do, its because the business meaning has changed. Like if a client code field needs to be increased, its because we bought out another company or something and need to prefix a division in there.

What irritates me most is not the slowness of the resizing, but this attitude that some people have that just because some of us see the value of limiting sizes of fields, some time machine from the cave ages must have accidentally dropped us off in their century and we need to be redumacated :)

CN Liou
This is exact one of my two reasons for which I use varchar(n) instead of text:

"The size gives auto form generators a sense of the width to make a field and limits to set on what can be put in it."

Another reason is, as already mentioned by other folks, the ODBC-like interface component I use converts varchar and text to different user interface properties.

the6campbells
Problems with the implementation.

1. char() not preserving space - yes docmented but catches people out who ue multiple vendors

2. expressions which come back as text and often in ODBC drivers as a long varchar which to a dynamic sql engine is essentially a

lob. eg. select 'A' || C1 || 'Z' from T and so on

Some 'applicance' vendors who have postres in their 'roots' seem to have changed their engines to not carry this idiom forward but a few still have it.

Similar comments if you want to compare ANSI/ISO interval and timestamp issues re TZ which can catch folks out.

Import fixed width data into PostgreSQL with just PSQL

dim

Or try using pgloader which support fixed file format.

See http://pgloader.projects.postgresql.org/ and more precisely http://pgloader.projects.postgresql.org/#toc9

PostGIS Raster its on: 10 things you can do NOW with raster

BostonGIS Blog

Just finished writing the first draft of the last chapter of PostGIS in Action title First look at PostGIS WKT Raster. We now have in place drafts of all the chapters we planned to write. It was probably the most fun chapter to write of all and was lon

www.dbrunas.com.ar

uberVU - social comments

This post was mentioned on Twitter by planetpostgres: Leo Hsu and Regina Obe: PostGIS Raster its on: 10 things you can do NOW with raster http://bit.ly/dpUkX9
