Postgres OnLine Journal: April 2008

An in-depth Exploration of the PostgreSQL Open Source Database

Table Of Contents

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

PostGIS and YUM

PostGIS 1.3.3 is out

PostGIS 1.3.3 has been released and is already in the PostgreSQL RPM and soon will be in Yum repository. PostgreSQL 8.3 users who are using PostGIS are encouraged to upgrade because this release contains a fix for a major bug that affected spatial aggregates in 8.3. The windows Application Stack Builder version with the update should be out within this or next week.

Shp2PgSQL Loader now can load plain DBFs

We've been working on the 2007 Topologically Integrated Geographic Encoding and Referencing (Tiger) data recently. For those who are familiar with the US Census current Tiger format, this is the first version to be released in Environmental Systems Research Institute (ESRI) Shape file format. We ran into one small problem. The Tiger data includes related data with no geometry. These come as plain old DBase (DBF) files. Prior versions of Shp2PgSQL could not deal with DBF files with no corresponding Shape (SHP) geometry files, but the version packaged with 1.3.3 can.

I would like to thank Paul Ramsey for checking over my DBF-only patch and squeezing it into this release. Now that I have gotten my hands dirty again with C code, I almost feel like a real programmer. As a side note, even if you don't use PostGIS, this should come in handy for loading any DBF file into PostgreSQL.

New PostgreSQL Yum Repository

We recently had the pleasure of trying out the new PostgreSQL YUM repository for Fedora/RedHat Enterprise/Cent OS distros that is maintained by Devrim GÜNDÜZ. It made the process of installation on Redhat Enterprise Linux a lot simpler.

In this issue we shall provide step by step instructions on using it geared toward the non-Red Hat Linux/CentOS indoctrinated folk (AKA the misguided Microsoft Windows people). The reason we feel this is necessary is that a lot of people develop on Windows and then deploy on Linux. The Linux experience can be somewhat intimidating, so we hope to make this an easier process by assuming you know little if anything about Linux. So stay tuned for that article.

Back to Table Of Contents

Constraint Exclusion when it fails to work *Intermediate*

What is Constraint Exclusion?

Constraint Exclusion is a feature introduced in PostgreSQL 8.1 which is used in conjunction with Table Inheritance to implement table partitioning strategies. The basic idea is you put check constraints on tables to limit what kind of data can be inserted into it. Constraint Exclusion will then in theory skip over tables whose check constraints guarantee there is no way for it to satisfy the condition of a query.

Constraint Exclusion is a great thing, but it has some limitations and quirks one needs to be aware of that in some cases will prevent it from kicking in. Below are a couple of reasons why it sometimes doesn't work.

How many ways can constraint exclusion fail - let us count the ways

- Its disabled by default. You need to either change your **postgresql.conf** to constraint_exclusion = on. You can also enable it on a session by session basis. Alternatively if you don't want it enabled for all your databases but don't want to always have to toggle it on, you can enable it per database with a command
  ```
  ALTER DATABASE somedb SET constraint_exclusion=on;
  ```
- Constraint_Exclusion only works for range or equality check constraints. This means that if you add a check constraint such as
  ```
  ALTER TABLE product_items_j ADD CONSTRAINT chk_item_name CHECK (item_name LIKE 'J%');
  ```

  The above constraint will never come into play in constraint exclusion. However if you wrote the constraint such as
  ```
  ALTER TABLE product_items_j ADD CONSTRAINT chk_item_name CHECK (item_name BETWEEN 'J' AND 'JZ');
  ```
  Then your constraint **may** be used in constraint exclusion scenario.

- Even if you have a range check constraint and your where condition should rightly be constrained, it often is not if your WHERE condition is not written in a range like manner. For example
  ```
  SELECT item_name FROM product_items WHERE item_name LIKE 'K%'
  ```

  will not be excluded by the above check constraint, however
  ```
  SELECT item_name FROM product_items WHERE item_name = 'Kettle'
  ```
  will be excluded by the above constraint.

How do you know that Constraint Exclusion is actually working?

You can tell that constraint exclusion is or is not working by looking at the explain plan. The PgAdmin III Visual Explain plan, we find even easier to detect this since its easy to spot tables that shouldn't be there by looking at the pretty visual explain plan. We covered this in Reading PgAdmin Graphical Explain Plans.

For completeness and ease of study below i is a simple exercise along with PgAdmin visual explain plan diagrams.

```
CREATE DATABASE playfield;
ALTER DATABASE playfield SET constraint_exclusion=on;

CREATE TABLE products
(
  item_id serial NOT NULL,
  item_name character varying(50) NOT NULL,
  CONSTRAINT pk_products PRIMARY KEY (item_id),
  CONSTRAINT uidx_products_item_name UNIQUE (item_name)
);

CREATE TABLE products_a_j
(
  CONSTRAINT pk_products_a_j PRIMARY KEY (item_id),
  CONSTRAINT uidx_products_a_j_item_name UNIQUE (item_name),
  CONSTRAINT chk_in_range_item_name CHECK (item_name >= 'A' AND item_name < 'K')
)
INHERITS (products);

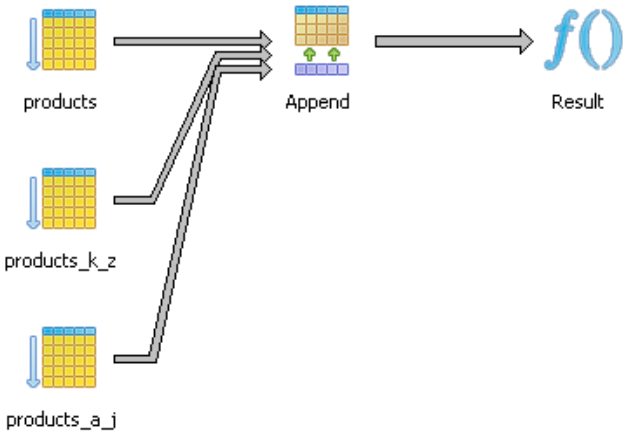
CREATE TABLE products_k_z
(
  CONSTRAINT pk_products_k_z PRIMARY KEY (item_id),
  CONSTRAINT uidx_products_k_z_item_name UNIQUE (item_name),
  CONSTRAINT chk_in_range_item_name CHECK (item_name >= 'K')
)
INHERITS (products);
```

```
--this is 8.2+ multi-insert syntax
INSERT INTO products_a_j(item_name)
VALUES ('APPLE'),('APRICOT'), ('AVOCADO'), ('JACKFRUIT'), ('GUAVA');

INSERT INTO products_k_z(item_name)
VALUES ('KOBE BEEF'),('WATERMELON'), ('LEMON');
```

```
---Test NO Constraint exclusion happening --
SELECT * FROM products
WHERE item_name LIKE 'A%';
```

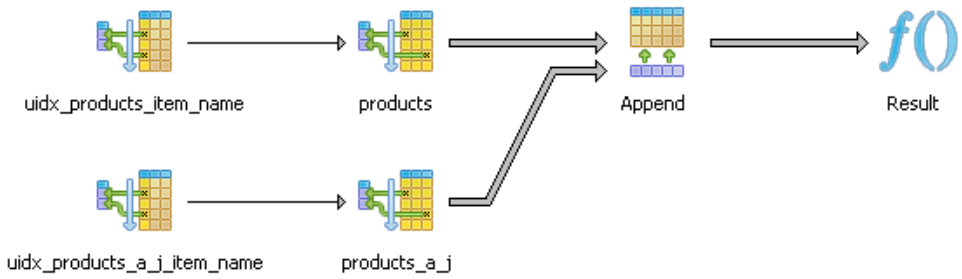
As we see in the below diagram - both child tables are being checked although there is no way the k_z table can satisfy this query.



Now if we rewrite the above query as one of the below examples:

```
--Rewrite query to use constraint exclusion --
SELECT * FROM products
WHERE item_name BETWEEN 'A' AND 'AZ';

--Another example demonstrating combo
-- and constraint exclusion works here too --
SELECT * FROM products
WHERE item_name < 'B' AND item_name LIKE 'A%';
```



We observe that our K-Z table falls out of our planner equation nicely.

Conclusion: Constraint Exclusion is nice, but it could be better

As shown above, constraint exclusion is fairly naive and when writing queries you have to take this into consideration. To use effectively when you have non-simple range conditions, you sometimes have to throw in a redundant condition to force it to kick in.

This makes it not easily usable in cases such as spatial partitioning of data with bounding boxes or more complex check conditions. In theory constraint exclusion can work nicely and more conveniently in many additional cases with some (hopefully minor) improvements of how it equates conditions. Hopefully we'll see some of these improvements in 8.4 or 8.5.

Back to Table Of Contents

How do you rename a database ***Beginner***

Problem:

You've created a database but made an embarrassing typo in the name or for whatever reason you don't like it. How do you rename this database?

Solution:

If you are using PgAdmin III, you will not see this option. Just one of the ways PgAdmin III lets us down. However there is a simple way of doing it with a PostgreSQL command which has been in existence even in the 7.4 days of PostgreSQL which is documented in PostgreSQL official docs on ALTER DATABASE. In order to do it, you need to first make sure everyone is out of the database (including yourself) otherwise you'll get an annoying *database is being accessed by other users* or *current database may not be renamed* error.

1. Connect to some other database other than the one you are trying to rename such as say the **postgres** db.
2. Kick everyone out of the database you are trying to rename - to figure out users, you can run

```
SELECT *
        FROM pg_stat_activity
        WHERE datname = 'myolddbname_goes_here'
```

3. Now just run this command -
```
ALTER DATABASE myolddbname_here RENAME TO mynewdbname_here
```

Back to Table Of Contents

How to SELECT ALL EXCEPT some columns in a table ***Beginner***

People have asked us on several occasions if there is such a construct
`SELECT * EXCEPT(...list) FROM sometable`
. Sadly we do not think such a thing exists in the ANSI SQL Specs nor in PostgreSQL.

The above feature would come in handy when you have certain fields in your tables that are common across tables, but you need to leave them out in your query. A common case of this is when you have PostGIS tables loaded using *shp2pgsql* with a fields called gid and the_geom which are not terribly useful for simple data queries.

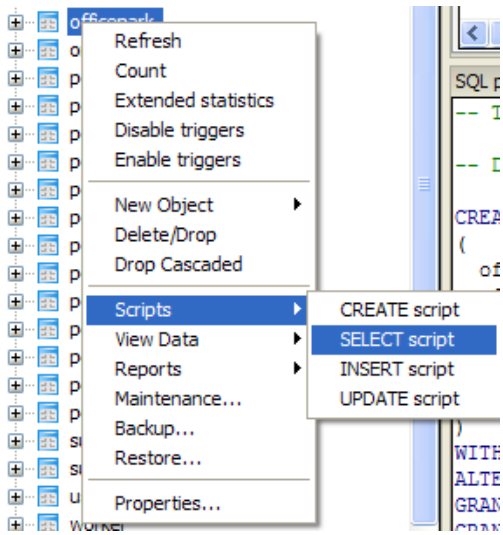
There are 2 common ways we use to achieve this result.

- Using PgAdmin's CREATE SELECT script feature. This exists in other GUI tools as well.
- Using an Information Schema script hack to construct the SELECT statement

The PgAdmin III way

Here are the steps:

1. Select the table you want from the tree.
2. Right mouse-click
3. Select Scripts->**SELECT script**
4. This should pop up a SELECT .. script with all columns selected. Now just cut out the fields you don't want.



The Information Schema Hack Way

```
SELECT 'SELECT ' || array_to_string(ARRAY(SELECT 'o' || '.' || c.column_name
        FROM information_schema.columns As c
            WHERE table_name = 'officepark'
            AND  c.column_name NOT IN('officeparkid', 'contractor')
    ), ',') || ' FROM officepark As o' As sqlstmt
```

The above for my particular example table - generates an sql statement that looks like this

```
SELECT o.officepark,o.owner,o.squarefootage
    FROM officepark As o
```

Back to Table Of Contents

Basics

An Almost Idiot's Guide to PostgreSQL YUM *Beginner*

First we'd like to thank Devrim of Command Prompt for working hard on making this new YUM repository available. In this article we will go over using the new PostgreSQL YUM repository for Redhat Fedora, Enterprise Linux and CentOS distros that is available at http://yum.pgsqlrpms.org/.

We are gearing the content of this article to the described user profile

- Person has SSH root access to their PostgreSQL box
- Person is new to using Red Hat Linux
- Person may be new to using Linux in general

Steps before getting started

1. SSH into your Linux box. For windows users, you can use Putty which we covered in PuTTY for SSH Tunneling to PostgreSQL Server
2. Log in as root
3. Determine which version of Linux you are running with the following commands:
   ```
   uname –a
   ```
   If this returns something with el4 (then you are running Enterprise Linux 4), el5 (Enterprise Linux 5), centos (ELsmp). Also pay attention to the bit 32-bit or 64-bit. 64bit will generally have an x64 and 32-bit will have i386 and/or i686 (for intel based).
   ```
   vi /etc/redhat-release
   ```
   also gives details of the version you are running

Backing up Old Version

If you are running a prior version of PostgreSQL, most likely it was installed in the /usr/local/pgsql/ folder. The first thing you want to do if you care about your data is to back it up with commands similar to the below.

1. ```
   mkdir dbbackup
   cd dbbackup
   /usr/local/pgsql/bin/pg_dumpall –U postgresqlserverdbs.sql
   ```
   You may also want to download the backup if its really important to you and in case you screw up the server really badly.
2. Now shut down the postgresql service

   ```
   su postgres
   /usr/local/pgsql/bin/pg_ctl stop -D /usr/local/pgsql/data
   ```

3. For extra good measure rename the old folder with and backup the postgresql.conf and pga_hba.conf to a safe location
   ```
   mv /usr/local/pgsql /usr/local/pgsqlold
   ```

Installing new Version

1. Login as root
2. Follow the instructions at http://yum.pgsqlrpms.org/howtoyum.php to prevent your YUM update from getting postgresql from other sources
3. Select the appropriate repository config file for your OS and choose 8.3 from here and navigating thru: http://yum.pgsqlrpms.org/reporpms/repoview/letter_p.group.html
4. Note the install file - should look something like http://yum.pgsqlrpms.org/reporpms/8.3/pgdg-redhat-8.3-4.noarch.rpm
5. Do a wget of the appropriate one: e.g.
   ```
   wget http://yum.pgsqlrpms.org/reporpms/8.3/pgdg-redhat-8.3-4.noarch.rpm
   ```
6. Next install the rpm config file with:
   ```
   rpm -ivh pgdg-redhat-8.3-2.noarch.rpm
   ```
7. Install the things that you want. These are the ones we tend to install

   ```
   yum install postgresql
   yum install postgis
   yum install pgadmin3
   yum install postgresql-contribs
   ```

8. We tend to like PostgreSQL in the /usr/local/pgsql/data folder since thats where we are used to finding it. So we init there. The default location of new install is different for each system. On EL its /var/lib/pgsql/data so you may want to init there although why its there feels so counter intuitive. Consider the below an example use case.
   ```

```
mkdir /usr/local/pgsql
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su postgres
initdb –D /usr/local/pgsql/data
```

9. `cd /usr/local/pgsql/data`

   and edit the postgresql.conf and pg_hba.conf files to your hearts content. It is fairly safe to copy over the pg_hba.conf file from your backup (using e.g
   `cp /usr/local/pgsqlold/data/pg_hba.conf /usr/local/pgsql/data`

) but for the postgresql.conf file, we would work with the new and cut in changes from the old since new things have been added between 8.2 and 8.3

Having PostgreSQL start as a service

The PostgreSQL install puts in a postgresql service script that you can tweak for your specific need. On RHEL 5, its located in /etc/rc.d/init.d. For other installs, it varies

1. `emacs /etc/rc.d/init.d/postgresql`

   Note: if you don't have emacs, you can use vi, but I tend to prefer emacs for simple edits.
2. Replace all references of /var/lib/pgsql/data with whereever you initD your database cluster
3. In emacs **Ctrlx+Ctrlc** to save
4. `service postgresql start`

   to test to make sure the service can start
5. Next to make sure it starts automatically on boot - do the following command

```
chkconfig --list (to see list of services)
               chkconfig postgresql on
```

   the chkconfig on thing automatically copies symlinks to those rc.1 , rc.n locations that linux looks for in bootup. If you don't have chkconfig on your box for some reason, you have to manually copy those symlinks to the right locations - a pain.

Installing PgAdmin pack

The PgAdmin pack comes with PgAdmin3. This comes in handy if you use PgAdmin3 a lot as it gives you stats activity, and allows you to change pg_hba.conf and postgresql.conf files directly from PgAdmin3 even if accessing from a different computer

```
psql -U postgres -d postgres -f /usr/share/pgsql/contrib/adminpack.sql
```

Back to Table Of Contents

Using DbLink to access other PostgreSQL Databases and Servers *Intermediate*

People coming from SQL Server and MySQL often complain about how you can't query other databases from within a PostgreSQL database. In Microsoft SQL Server, there is a concept of querying across databases on the same server with dbname.dbo.sometable and querying across servers (even of different types e.g. Oracle) by setting up a linked server and doing something such as servername.dbname.dbo.sometable or using the OPENROWSET(..) or OPENQUERY(...) syntax.

MySQL has a similar feature by using syntax dbname.sometable, but MySQL lacks schemas so there is no way to segregate a database into nice buckets as you can with SQL Server and PostgreSQL.

In this article we shall provide some examples of using the contrib module - **dblink** to query local PostgreSQL databases and remote PostgreSQL databases. DbLink is probably most comparable in structure to SQL Server's OpenRowset functionality. It lacks the power of SQL Server's Linked Server approach or OPENQUERY that allows for synchronized joins between linked servers/databases and local tables and updates/inserts on linked servers. This makes it not terribly useful in cases where you need to join lots of data with local data. It does however come in handy for bulk copy operations from one database/server to another.

Install DbLink

DbLink is located in share/contribs/dblink.sql of your PostgreSQL install.To use it, load the dblink.sql file in the database you would like to use it in.

Querying other local PostgreSQL Databases with DbLink

DBLink returns a generic data row type. One of the annoying things about this is that in order to use it, you need to specify the output structure. Perhaps in future versions of PostgreSQL, this limitation of lack of introspection of generic row types will be irradicated. The lack of introspection is an issue not only for DbLink, but for other generic row type returning functions such as TableFunc cross tab functions.

Below is an example of querying a database on the same server and cluster using DbLink. Note if no username and password is specified, then DbLink connects with whatever account you are currently using.

```
SELECT towns.*
FROM dblink('dbname=somedb','SELECT town, pop1980 FROM towns')
   AS towns(town varchar(21), pop1980 integer);
```

If you are running on a non-standard port (something other than 5432), then you will need to specify the port as follows:

```
SELECT p.*
FROM dblink('dbname=ma_geocoder port=5433',
         'SELECT gid, fullname, the_geom_4269
               FROM ma.suffolk_pointlm')
   AS p(gid int,fullname varchar(100), the_geom geometry);
```

Querying Remote servers with DbLink and Bulk Insert

DBLink comes in particularly handy for querying remote PgServers and doing bulk inserts. To do so, you will need to specify the full credentials of the remote server. Below is an example of this.

```
SELECT blockgroups.*
   INTO temp_blockgroups
   FROM dblink('dbname=somedb port=5432
         host=someserver user=someuser password=somepwd',
   'SELECT gid, area, perimeter, state, county,
         tract, blockgroup, block,
         the_geom
         FROM massgis.cens2000blocks')
         AS blockgroups(gid int,area numeric(12,3),
            perimeter numeric(12,3), state char(2),
            county char(3), tract char(6),
            blockgroup char(1),
            block char(4), the_geom geometry);
```

Joining with Local Data

This is the major area where DbLink falls short. While you can join remote dblinked tables with local tables, for large tables, this tends to be really slow as indexes are not used and DBLinked function call needs to return the full dataset before they can be joined. Below is a simple example of a join:

```
SELECT realestate.address, realestate.parcel, s.sale_year, s.sale_amount,
      FROM realestate INNER JOIN
            dblink('dbname=somedb port=5432 host=someserver
            user=someuser password=somepwd',
            'SELECT parcel_id, sale_year,
               sale_amount FROM parcel_sales')
         AS s(parcel_id char(10),sale_year int, sale_amount int)
          ON realestate.parcel_id = s.parcel_id;
```

If sales table is relatively huge, then you may be better forcing a nested join by creating a function that queries dblink and putting that in the SELECT clause and having a dynamic dblink query of the form.

```
CREATE TYPE payment AS
   (
    payment_id integer,
    customer_id smallint,
    staff_id smallint,
    rental_id integer,
    amount numeric(5,2),
    payment_date timestamp);

CREATE OR REPLACE FUNCTION fn_remote_payments(cust int)
  RETURNS SETOF payment AS
$$

 SELECT * FROM dblink('dbname=pagila host=localhost user=pagila',
    'SELECT payment_id, customer_id, staff_id,
          rental_id, amount, payment_date
       FROM payment WHERE customer_id = ' || $1) As
          p(payment_id int,customer_id smallint,
             staff_id smallint,
          rental_id integer, amount numeric(5,2),
          payment_date timestamp )

$$
  LANGUAGE 'sql' VOLATILE;

SELECT rs.customer_id, realestate.address,
    realestate.parcel, (fn_remote_payments(rs.customer_id)).*
       FROM realestate rs
       WHERE rs.last_name = 'SMITH';
```

Potential Improvements

DBLink is not as feature rich as something like Microsoft SQL Server Linked Server or Oracle's DbLink. We can think of 2 features which hopefully will be implemented in future versions of PostgreSQL that will improve DbLink among other contribs.

- **Introspection:** As mentioned earlier in this article. This would allow you to do things like SELECT p.* FROM dblink('...', 'SELECT a, b,c FROM sometable') p without having to specify the output structure of the table. NOTE: That SQL Server supports this syntax with OPENROWSET, OPENQUERY and linked servers where you can do
  `SELECT p.* FROM OPENROWSET(connectionstring, 'SELECT a,b,c FROM sometable') p`

  or
  `SELECT p.* FROM remoteserver.remotedb.remoteschema.remotetable p`
- **Pipelining:** This is the idea of a set returning function returning values before the full set of data is materialized. We are not sure how helpful this would be but could conceivably help with creating synchronized joins.

Back to Table Of Contents

REST in PostgreSQL Part 2 A - The REST Server service with ASP.NET *Intermediate*

Setting up the .NET application

1. Download the npgsql 1.01 driver from pgfoundary - http://pgfoundry.org/projects/npgsql. For ASP.NET 2.0 you'll want - Npgsql1.0.1-bin-ms2.0.zip and for Mono.NET you'll want Npgsql1.0.1-bin-mono-2.0.zip. Unzip and place the files in bin folder of your web app project.

2. Since we are just creating a simple REST web service and don't need any plumming of the standard SOAP like webservice, we will be using a .NET handler class (ashx) instead of an asmx. We have two versions listed below. One for C# and one for VB.NET/Mono Basic

The following application files were tested on both Microsoft ASP.NET 2.0 IIS and Mono.NET 1.2.6 XSP Test Server ASP.NET 2

Below is the web.config file that works on both .NET implementations.

```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
  <connectionStrings>
   <add name="DSN"
      connectionString="Server=127.0.0.1;Port=5432;User Id=pagila_app;Password=pg@123;Database=pagila;"/>
  </connectionStrings>
   <system.web>
      <compilation debug="true" />
   </system.web>
</configuration>
```

Our VB.NET/Mono Basic Rest Service looks like this

'--PagilaSearch_VB.ashx

```
<%@ WebHandler Language="VB" Class="PagilaSearch_VB" %>
Imports System
Imports System.Web
Imports Npgsql

Public Class PagilaSearch_VB : Implements IHttpHandler
   Public Sub ProcessRequest(ByVal context As HttpContext) Implements IHttpHandler.ProcessRequest
      Dim maxrecs, offset As Integer
      maxrecs = 30 :  offset = 1
      If (Not IsNothing(context.Request("query"))) Then
         context.Response.ContentType = "text/xml"
         If IsNumeric(context.Request("maxrecs")) Then
            maxrecs = context.Request("maxrecs")
         End If
         If IsNumeric(context.Request("offset")) Then
            offset = context.Request("offset")
         End If
         context.Response.Write(GetResults(context.Request("query"), maxrecs, offset))
      End If
   End Sub

   Public ReadOnly Property IsReusable() As Boolean Implements IHttpHandler.IsReusable
      Get
         Return False
      End Get
   End Property

   Public Function GetResults(ByVal aquery As String, ByVal maxrecs As Integer, ByVal offset As Integer) As String
      Dim result As String = ""
      Dim command As NpgsqlCommand
      Using conn As NpgsqlConnection = New NpgsqlConnection( _
```

```
        System.Configuration.ConfigurationManager.ConnectionStrings("DSN").ConnectionString)

        conn.Open()
        command = New NpgsqlCommand("SELECT fnget_film_search_results(:search_criteria, :maxrecs, :offset)",
conn)
        command.Parameters.Add(New NpgsqlParameter("search_criteria", System.Data.DbType.String, 300)).
Value = aquery
        command.Parameters.Add(New NpgsqlParameter("maxrecs", System.Data.DbType.Int16)).Value = maxrecs
        command.Parameters.Add(New NpgsqlParameter("offset", System.Data.DbType.Int16)).Value = offset

        Try
            result = command.ExecuteScalar()
        Catch ex As Exception
            Return "<error>ERROR " & ex.ToString() & "</error>"
        End Try
    End Using
    Return result
  End Function
End Class
```

CSharp REST Service

Our CSharp equivalent looks like this

```
<%@ WebHandler Language="C#" Class="PagilaSearch_CS" %>
using System;
using System.Web;
using Npgsql;

public class PagilaSearch_CS : IHttpHandler {

  public void ProcessRequest (HttpContext context) {
    int maxrecs = 30, offset = 1;
    if (context.Request["query"] != null){
      context.Response.ContentType = "text/xml";
      if (context.Request["maxrecs"] != null) {
        try {
          maxrecs = Convert.ToInt16(context.Request["maxrecs"].ToString());
        }
        catch {//ignore the exception
        }
      }

      if(context.Request["offset"] != null) {
        try {
          offset = Convert.ToInt16(context.Request["offset"].ToString());
        }
        catch { //ignore the exception
        }
      }
      context.Response.Write(GetResults(context.Request["query"], maxrecs, offset));
    }
  }

  public bool IsReusable {
    get {
      return false;
    }
  }

  public String GetResults(String aquery, int maxrecs, int offset){
    String result = "";
    NpgsqlCommand command;
    using (NpgsqlConnection conn = new NpgsqlConnection(
        System.Configuration.ConfigurationManager.ConnectionStrings["DSN"].ConnectionString))
    {
      conn.Open();

      command = new NpgsqlCommand("SELECT fnget_film_search_results(:search_criteria, :maxrecs, :offset)",
conn);
      command.Parameters.Add(new NpgsqlParameter("search_criteria", System.Data.DbType.String, 300)).Value
= aquery;
      command.Parameters.Add(new NpgsqlParameter("maxrecs", System.Data.DbType.Int16)).Value = maxrecs;
      command.Parameters.Add(new NpgsqlParameter("offset", System.Data.DbType.Int16)).Value = offset;
```

```
        try
        {
           result = (String)command.ExecuteScalar();
        }
        catch(Exception ex)
        {
           return "<error>ERROR " + ex.ToString() + "</error>";
        }
     }
     return result;
   }

}
```

Running in Mono

To run on Mono -

- Install Mono by picking respective install from here - http://www.go-mono.com/mono-downloads/download.html. **NOTE:** There is a newer version than what I was using, but should work fine with the newer version as well.
- Create a folder called pagila in the lib/xsp/2.0 folder
- throw the files in the folder
- launch the web server

Testing our REST Service

Now to test our service - we shall try searching for films *not about dentists and not about dogs but involving mad nigeria*

Our search phrase is *not dentist not dog mad nigeria* and our browser call is like

```
http://localhost:8080/2.0/pagila/PagilaSearch_VB.ashx?query=not%20dentist%20mad%20not%20dog%
20nigeria&maxrecs=20&offset=1
```

```
http://localhost:8080/2.0/pagila/PagilaSearch_CS.ashx?query=not%20dentist%20mad%20not%20dog%
20nigeria&maxrecs=20&offset=1
```

The result of our REST query looks like this

```
<results>
 <resultsummary>
   <count>4</count>
</resultsummary>
<table xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance" xmlns="vwfilms">
 <row>
<fid>883</fid>
<title>TEQUILA PAST</title>
<description>A ActionPacked Panorama of a Mad Scientist
And a Robot who must Challenge a Student in Nigeria</description>
<category>Children</category>
<price>4.99</price>
<length>53</length>
<rating>PG</rating>
</row>
 <row>
<fid>614</fid>
<title>NAME DETECTIVE</title>
<description>A Touching Saga of a Sumo Wrestler
And a Cat who must Pursue a Mad Scientist in Nigeria</description>
<category>Games</category>
<price>4.99</price>
<length>178</length>
<rating>PG13</rating>
</row>
 <row>
<fid>34</fid>
<title>ARABIA DOGMA</title>
<description>A Touching Epistle of a Madman
And a Mad Cow who must Defeat a Student in Nigeria</description>
<category>Horror</category>
<price>0.99</price>
<length>62</length>
<rating>NC17</rating>
</row>
</table>
</results>
```

Back to Table Of Contents

REST in PostgreSQL Part 2 B - The REST Server service with PHP 5 *Intermediate*

This is a continuation of our REST series. The following topics have already been covered

1. Showcasing REST in PostgreSQL - The PreQuel we went over what REST is and isn't
2. REST in PostgreSQL Part 1 - The DB components we loaded the Pagila database and created a db plpgsql search function that spits out XML to support our rest server service
3. REST in PostgreSQL Part 2 A - The REST Server service with ASP.NET we demonstrated a REST web service using Mono.NET, MS.NET both in C#, VB.Net/Monobasic

Now in this Part 2B series, we shall demonstrate the same REST server service using PHP

Setting up the PHP application

1. PHP already has the PostgreSQL drivers available as a .so (on Linux) or .dll on Windows. For windows users if you are running PHP under IIS and in ISAPI mode, you will not be able to dynamically load libraries, so you need to enable **php_pgsql** in your PHP.ini file.

2. We tend to keep it enabled regardless of which platform we are on since a lot of our PHP development involves PostgreSQL. The extension is *php_pgsql* in the php.ini file
3. PHP has numerous database abstraction libraries to choose from. We are using the adodb abstraction library for PHP which can be downloaded from http://adodb.sourceforge.net/.

First we have a simple php config file that looks as follows.

--Contents of config.inc.php

```
<?php
   $CONNSTRING = "postgres://pagila_app:pg@123@localhost:5433/pagila?persist";
?>
```

And now for our php rest service. Keep in mind this is only one of millions of ways of doing this in PHP. The nice thing about PHP is that it gives you lots of freedom. The bad thing about PHP is that it gives you lots of freedom.

/**pagilasearch_php.php **/

```
<?php
include_once("config.inc.php");
include_once("adodb/adodb.inc.php");
include('adodb/adodb-exceptions.inc.php');
class PagilaSearch_PHP {
   function __construct() {
      $this->process_request();
   }

   function __destruct() { //do cleanup

   }

   protected function process_request() {
      $maxrecs = 30; $offset = 1;
      if (!empty($_REQUEST["query"])){
         header('Content-type: text/xml');
         if (is_numeric($_REQUEST["maxrecs"])) {
           $maxrecs = $_REQUEST["maxrecs"];
         }

         if(is_numeric($_REQUEST["offset"])) {
           $offset = $_REQUEST["offset"];
         }
         echo $this->get_results($_REQUEST["query"], $maxrecs, $offset);
      }
   }
```

```
  protected function get_results($aquery, $maxrecs, $offset){
     global $CONNSTRING;
     $conn =  &ADONewConnection($CONNSTRING);

     if (!$conn->PConnect()) {
        $result = "<error>ERROR Can not connect to db</error>";
     }
     else {
        $conn->SetFetchMode(ADODB_FETCH_ASSOC);
        $conn->debug = false;
     }
     try {$rs = $conn->Execute("SELECT fnget_film_search_results(" . $conn->qstr($aquery) . ", " .
            (int) $maxrecs . ", " . (int) $offset . ") As answer")->GetRows();

        $result = $rs[0]['answer'];
     }
     catch(Exception $e){
        $result = "<error>ERROR " . $e . "</error>";
     }

     return $result;
  }
}

new PagilaSearch_PHP();
?>
```

Testing our REST Service

Now to test our service - we shall try searching for films *not about dentists and not about dogs but involving mad nigeria*

Our search phrase is *not dentist not dog mad nigeria* and our browser call is like

```
http://localhost/pagilasearch_php.php?query=not%20dentist%20mad%20not%20dog%
20nigeria&maxrecs=20&offset=1
```

The result of our REST query looks like this

```
<results>
 <resultsummary>
   <count>4</count>
</resultsummary>
<table xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance" xmlns="vwfilms">
 <row>
<fid>883</fid>
<title>TEQUILA PAST</title>
<description>A ActionPacked Panorama of a Mad Scientist
And a Robot who must Challenge a Student in Nigeria</description>
<category>Children</category>
<price>4.99</price>
<length>53</length>
<rating>PG</rating>
</row>
 <row>
<fid>614</fid>
<title>NAME DETECTIVE</title>
<description>A Touching Saga of a Sumo Wrestler
And a Cat who must Pursue a Mad Scientist in Nigeria</description>
<category>Games</category>
<price>4.99</price>
<length>178</length>
<rating>PG13</rating>
</row>
 <row>
<fid>34</fid>
<title>ARABIA DOGMA</title>
<description>A Touching Epistle of a Madman
And a Mad Cow who must Defeat a Student in Nigeria</description>
<category>Horror</category>
<price>0.99</price>
<length>62</length>
<rating>NC17</rating>
</row>
</table>
```

```
</results>
```

Back to Table Of Contents

PuTTY for SSH Tunneling to PostgreSQL Server *Beginner*

What is PuTTY

PuTTY was developed by Simon Tatham and is a very common light-weight MIT-Licensed free and open source Secure Shell (SSH) client for connecting to Linux/Unix systems via a Teletype (TTY) terminal emulation mode console. Currently there are ports for Microsoft Windows, other unix like systems, and ports in progress for Mac OSX and Symbian mobile phone OS.

PuTTY fits into that class of tools we affectionately call *Swiss Army Knives* because it is Light, Multi-Purpose, and Good Enough. As an added benefit it is free and open source with a generous license so it is commonly embedded in commercial apps.

PuTTY comes in handy both as an SSH terminal console and as a SSH Tunneling tool which allows you for example to use PgAdmin III from a local windows workstation against a remote PostgreSQL server even in cases where the linux/unix PostgreSQL **pg_hba.conf** and **postgresql.conf** file only allow local connections or non-SSH traffic is blocked by firewall. For more about the nuances of configuring the pg_hba.conf PostgreSQL server file that controls user access check out Hubert Lubaczewski's "FATAL: Ident authentication failed", or how cool ideas get bad usage schemas http://www.depesz.com/index.php/2007/10/04/ident/

In this article we shall cover how to use PuTTY's SSH Tunneling feature to access a remote PostgreSQL server that doesn't allow remote connections. To make it a little more interesting we shall demonstrate how to do this for PgAdmin III.

SSH Tunneling and why you might need it

First off we'd like to say the command-line tool **psql** that also comes with PostgreSQL is nice and has its charm. It is simple enough that it can be run from an SSH console without tunneling so not much need for SSH Tunneling here. Unfortunately psql is scary to beginning PostgreSQL users, and also requires you have a fair number of SQL commands memorized to make the most use of it. PSQL also requires some typing which is annoying for many general use cases. It is ideal for scripting things and so forth, but it just is not a GUI app and was not designed to be.

This is where the PgAdmin III tool fits the bill. PgAdmin III comes packaged with PostgreSQL but can also be installed separately. Now how do you use PgAdmin III when all you have is shell access to your server box and all the PostgreSQL ports are blocked or pg_hba.conf is configured to not allow remote access? This is where SSH Tunneling comes in handy.

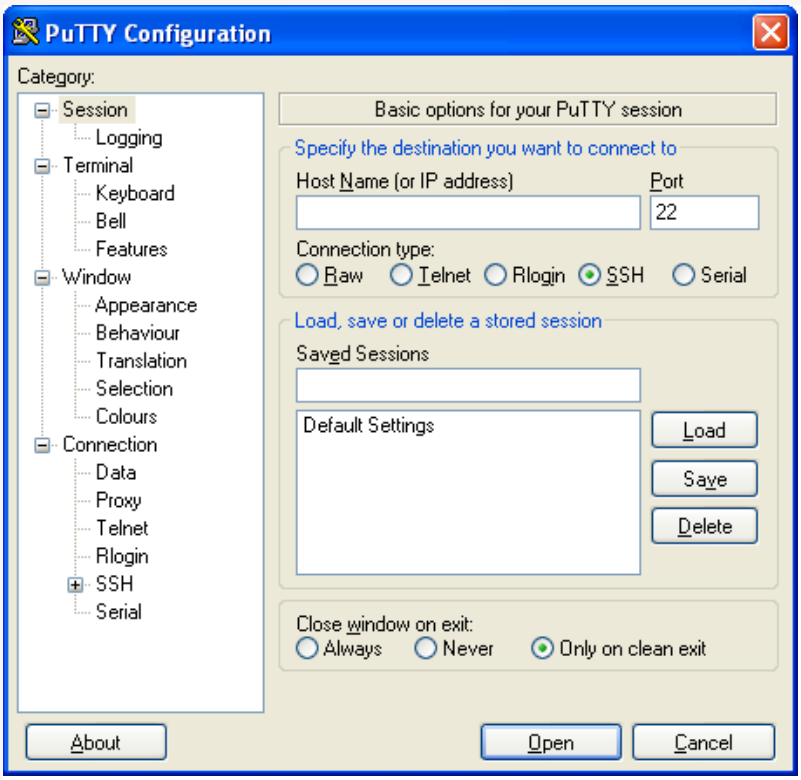
What SSH Tunneling allows you to do is to tunnel all your traffic to the server thru your SSH connection. It is basically a Virtual Private Network (VPN) using SSH. The basic idea is you map local ports on your pc to remote service ports on the server. When you launch your SSH session, you can then connect with any application e.g. PgAdmin III, MS Access whatever to this remote port via the local port. Instead of specifying the remote server port when setting up your PgAdmin III or MS Access connection, you specify the ip as localhost and port as whatever port you configured to receive traffic via the Tunnel.

Setting up SSH Tunneling with PuTTY

If you do not have PuTTY already, you can download it from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Windows users can download the respective putty.exe. Putty is fairly small and can even fit on a floppy. It is really sweet because it requires no installation - just click and run.

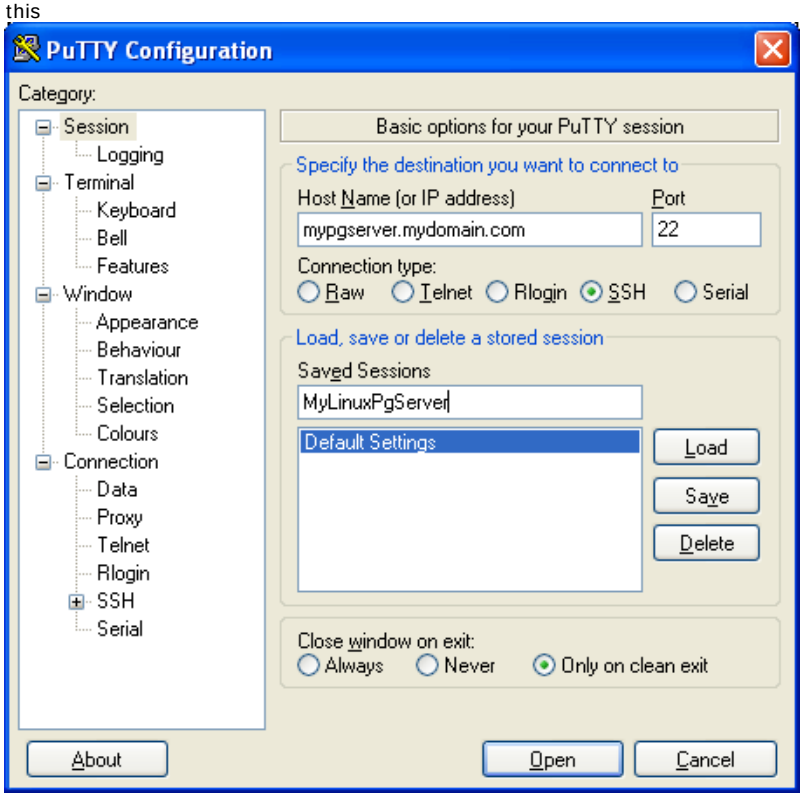
Once downloaded, simply launch the putty.exe.

Once launched, your screen will look something like this.

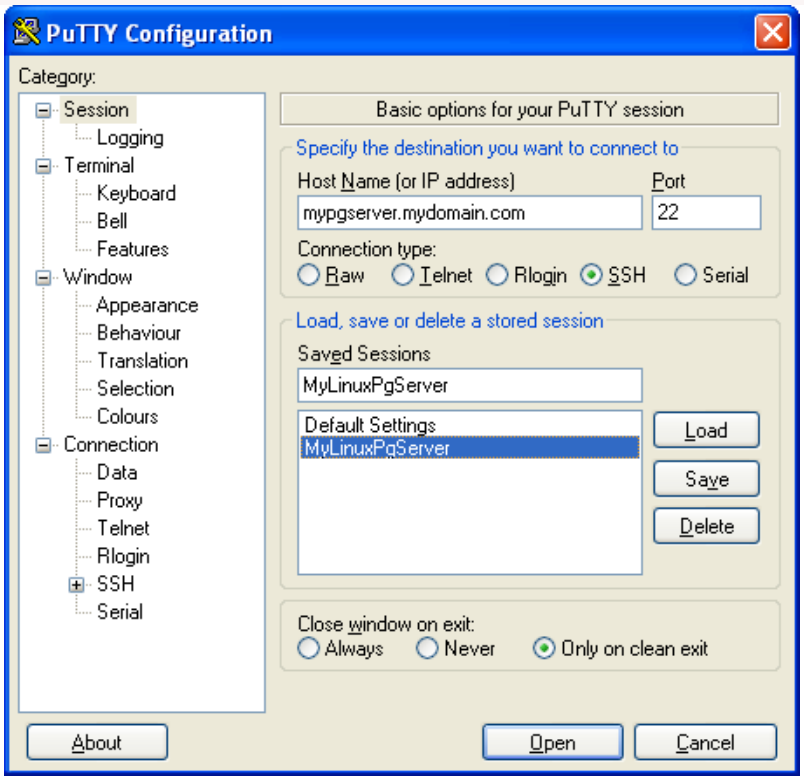


Now create a connection by following these steps:

- Fill in a hostname or ip address of your PostgreSQL Server in the field labeled **(Hostname or IP Address)**
- Fill in The name you'd like to reference this connection in the **Saved Sessions** field. At this point your screen should look something like this

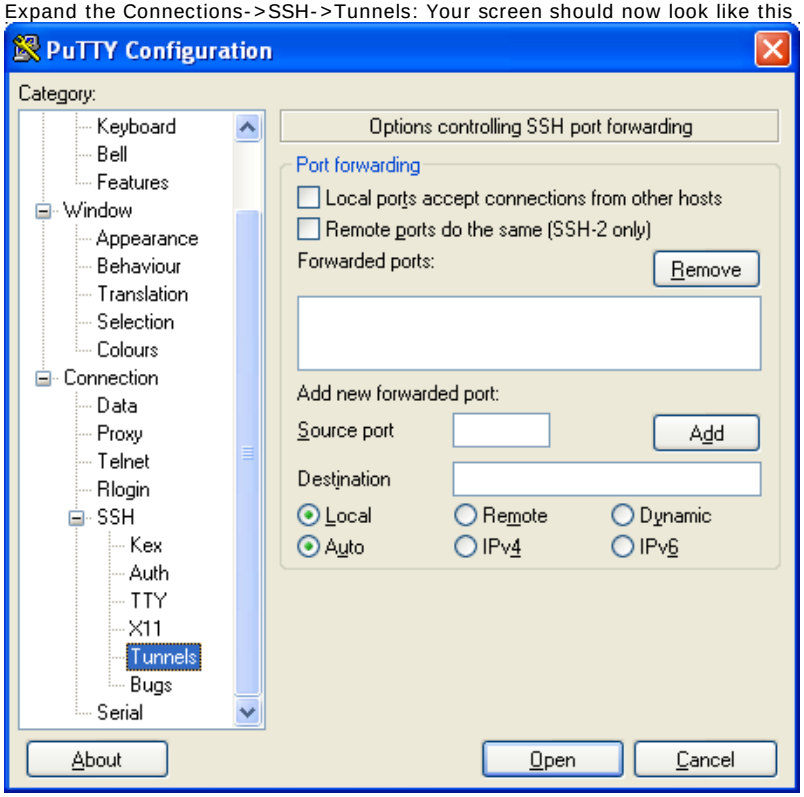


- Click the **Save** button, and select the new session - your screen should look like this:



Now set the tunneling configuration

- To set the tunneling configuration of this new session or an existing session, simply select session
- Click **Load**
- Expand the Connections->SSH->Tunnels: Your screen should now look like this

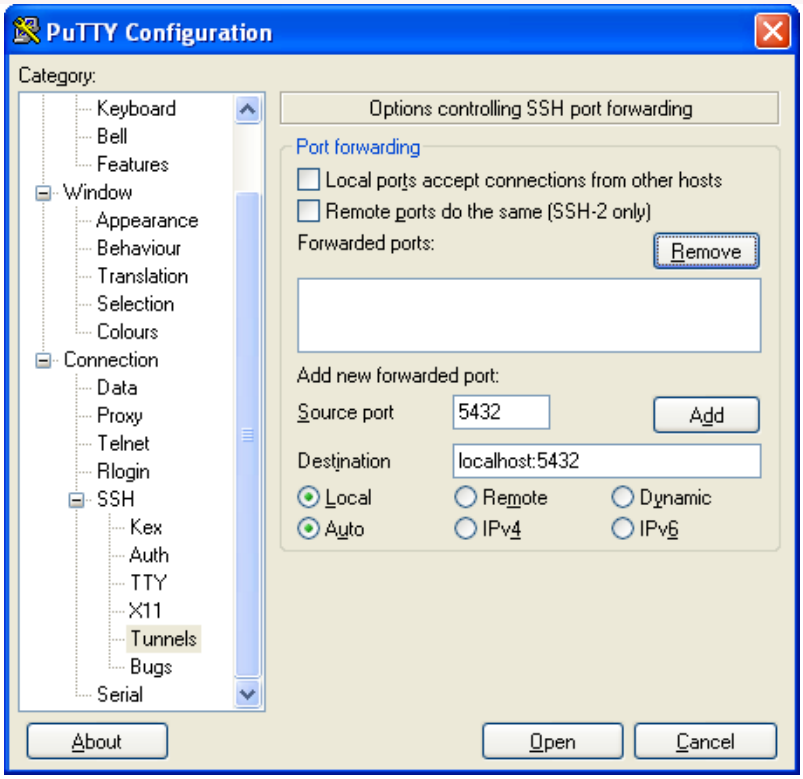


- Next click the **Local** radio box and type in the Source Port - this is the server's postgresql port which is generally **5432**

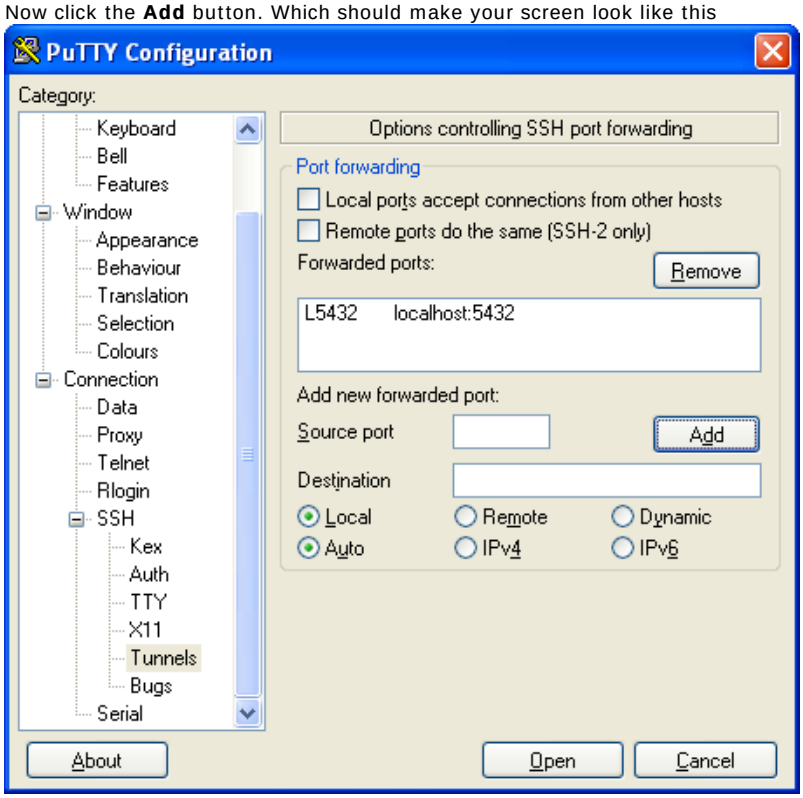
- In Destination - this is the server address and port you want the traffic redirected to. For personal desktop use, we tend to use localhost:5432 (if you are not running a postgresql dev server locally) or localhost:someotherunusedport (e.g. localhost:5433).

  Note: You could very well put the LAN ip address of your workstation in here such as 192.168.1.2 if you want local users to share your Tunnel connection.

  At this point your screen should look something like this



- Now click the **Add** button. Which should make your screen look like this



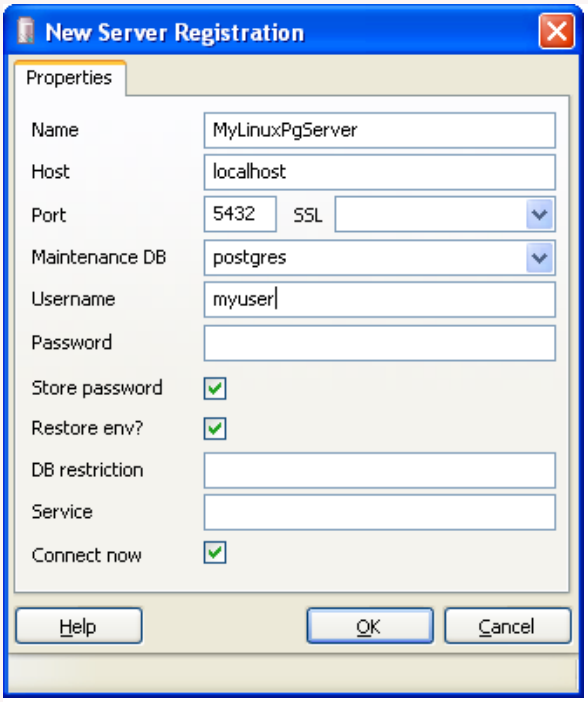
Keep in mind that although we are focusing on PostgreSQL, you can forward other Linux/Unix service traffic such as VNC, FTP etc by adding more entries

- Now go back to the Session section again and click the **Save** button and then click **Open** to Launch the Session.

Once you do all of the above you and your Linux/Unix server is enabled with SSH support, you should get a black login console. Login as usual and just keep the console running.

Connecting with PgAdmin III

Connecting with PgAdmin III is now simple. The only trick is that instead of using the server's name and port, you specify the destination you chose instead. In the above we had chosen localhost:5432 so we setup a PgAdmin III connection with that as shown below.



Back to Table Of Contents

Reader Comments

PostGIS and YUM

asd
It would fantastic to have Slony in the repository...

How do you rename a database

Fabrice De Volder
Hi,

My problem is to change the encoding of a running database without backup, modify and restore.

Any solution ?

Thanks for your journal, it's great.

Fabrice.

Leo
Fabrice,

Unfortunately I don't think there is a way to change the server encoding of a database without a dump reload, however you can change the default client encoding of a database without a dump restore (although not all client encodings are compatible with each server encoding). There is an automatic conversion from client to server encoding when they are compatible.

To change default client encoding of a database, you would do something like

ALTER DATABASE somedb SET client_encoding=latin1;

OR

ALTER DATABASE some SET client_encoding=utf8;

All the valid encodings are listed here
http://www.postgresql.org/docs/8.3/static/multibyte.html

How to SELECT ALL EXCEPT some columns in a table

David Fetter
This is a neat hack, and an interesting example of SQL-generating SQL.

One little thing to add: when somebody asks you for a thing like this, what they're really doing is expressing pain over a denormalized schema, so it's good to take a broader look at that part, first chance you get.

Regina
Good point. We do ask people first why they want to do that before we suggest a solution. Surprisingly a lot of people want to do this for one off things like this and use cases we never thought of.
I guess the point is just because you've never had a need for something, don't dismiss other people without hearing them out.

For us for example we have built custom query builders, where we want to allow admin users to query from all fields except for example big blob fields and so forth (e.g. geometry fields). This comes in handy in that regard too since you can exclude certain data types as well as named fields since the information_schema.columns table provides data type info.

Dricks
"SELECT * EXCEPT(...list) FROM sometable" :
Great syntax suggestion.
This actually is painfull to select all but some fields (as regina suggested, we are actually doing the exact same thing : exclude binary fields or large text fields).
Which means that we are actually doing it using two request : first one to get the columns of the table, then remove unwanted ones, then the real request.
With this, we could do all of this in 1 database access...
I would love that!

PuTTY for SSH Tunneling to PostgreSQL Server

Didier Bretin
Thanks for the demonstration. It's a very good tip.

Erik Jones
Excellent article. However, you may want to add another example covering the extremely common case where a user doesn't have any direct access to the db server, i.e they can log in to their web server and that login can be set up to forward connections to the db server.

Autumn
Did they every email you and show you how to create this link with a db server which doesn't have any direct access???
If so -can you forward me to the link???
thank you.

Postgres OnLine Journal
First we'd like to thank Devrim of Command Prompt for working hard on making this new YUM repository available. In this article we will go over using the new PostgreSQL YUM repository for Redhat Fedora, Enterprise Linux and CentOS distros that is availab