PostgreSQL 数据库

一:PostgreSQL 介绍

1.PostgreSQL 是以加州大学伯克利分校计算机系开发的 POSTGRES，现在已经更名为 POSTGRES，版本 4.2 为基础的对象关系型数据库管理系统（ORDBMS）。PostgreSQL 支持大部分 SQL 标准并且提供了许多其他现代特性：复杂查询、外键、触发器、视图、事务完整性、MVCC。同样，PostgreSQL 可以用许多方法扩展，比如， 通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语言。并且，因为许可证的灵活，任何人都可以以任何目的免费使用、修改、和分发 PostgreSQL，不管是私用、商用、还是学术研究使用。

2.PostgreSQL 图标

3.PostgreSQL 优点

有目前世界上最丰富的数据类型的支持支持,其中有些数据类型可以说连商业数据库都不具备,具体类型下文会说明.

PostgreSQL 拥有一支非常活跃的开发队伍，而且在许多黑客的努力下，PostgreSQL 的质量日益提高

PostgreSQL 对接口的支持也是非常丰富的，几乎支持所有类型的数据库客户端接口。这一点也可以说是 PostgreSQL 一大优点。

4.PostgreSQL 缺点

首先，早期的 PostgreSQL 继承了几乎所有 Ingres,Postgres,Postgres95 的问题：过于学院味，因为首先它的目的是数据库研究，因此不论在稳定性， 性能还是使用方方面面，长期以来一直没有得到重视，直到 PostgreSQL 项目开始以后，情况才越来越好，PostgreSQL 已经完全可以胜任任何中上规模范围内的应用范围的业务

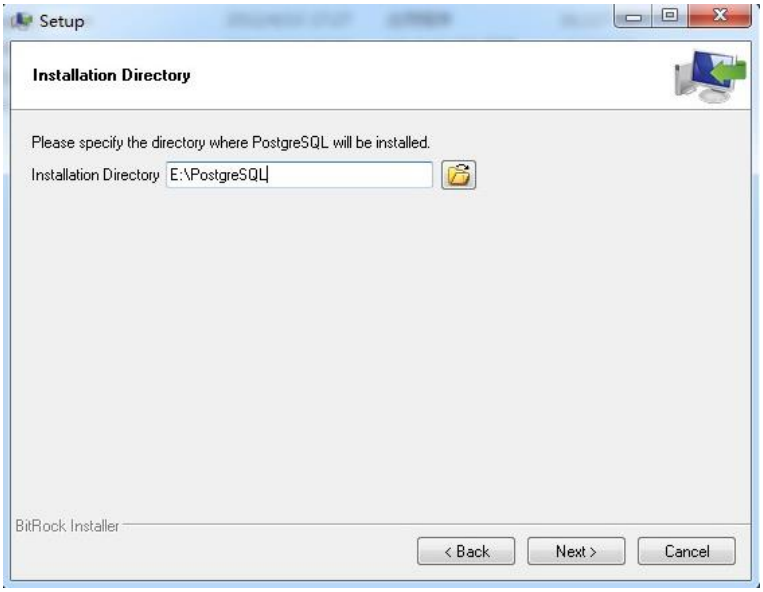
其次，PostgreSQL 的确还欠缺一些比较高端的数据库管理系统需要的特性，比如数据库集群，更优良的管理工具和更加自动化的系统优化功能 等提高数据库性能的机制等。

5.目前官方最新版本:9.3.2

二、windows 下安装过程

1、开始安装：

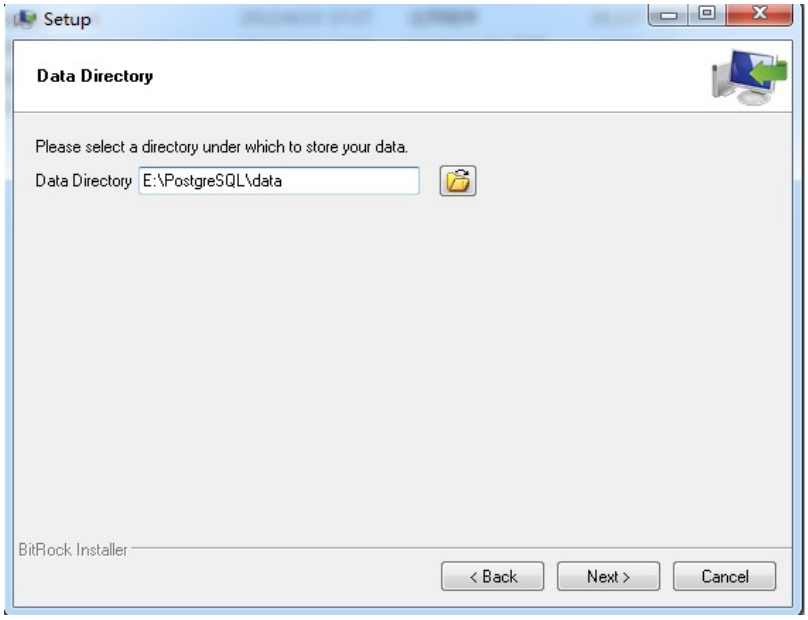
2、选择程序安装目录：



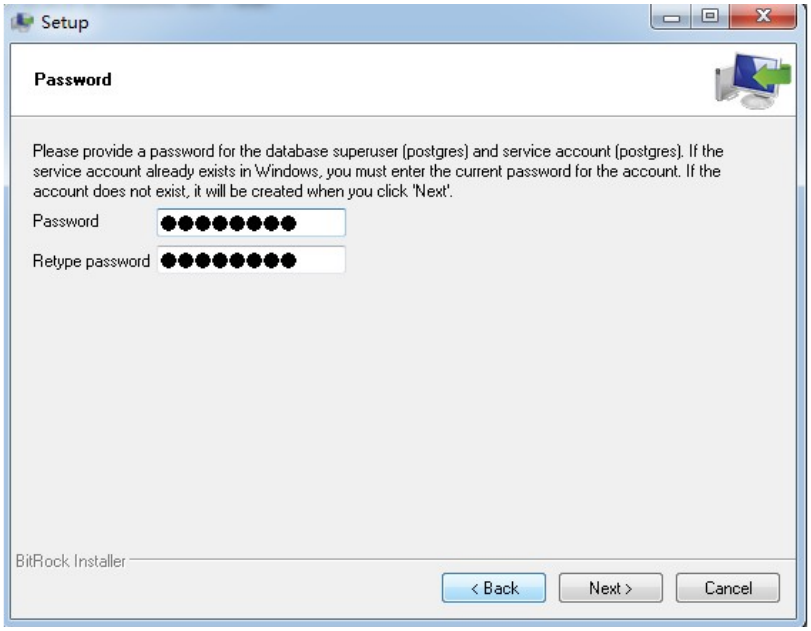
注：安装 PostgreSQL 的分区最好是 NTFS 格式的。PostgreSQL 首要任务是要保证数据的完整性，而 FAT 和 FAT32 文件系统不能提供这样的可靠性保障，而且 FAT 文件系统缺乏安全性保障，无法保证原始数据在未经授权的情况下被更改。此外，PostgreSQL 所使用的"多分点"功能完成表空间的这一特征在 FAT 文件系统下无法实现。

然而，在某些系统中，只有一种 FAT 分区，这种情况下，可以正常安装 PostgreSQL，但不要进行数据库的初始化工作。安装完成后，在 FAT 分区上手动执行 initdb.exe 程序即可，但不能保证其安全性和可靠性，并且建立表空间也会失败。

3、选择数据存放目录：

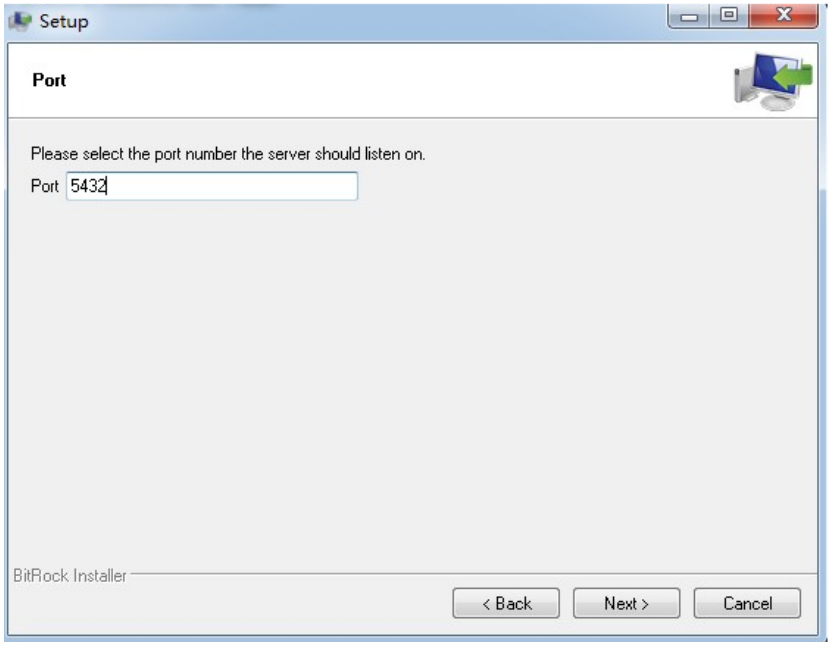


4、输入数据库超级用户和创建的 OS 用户的密码

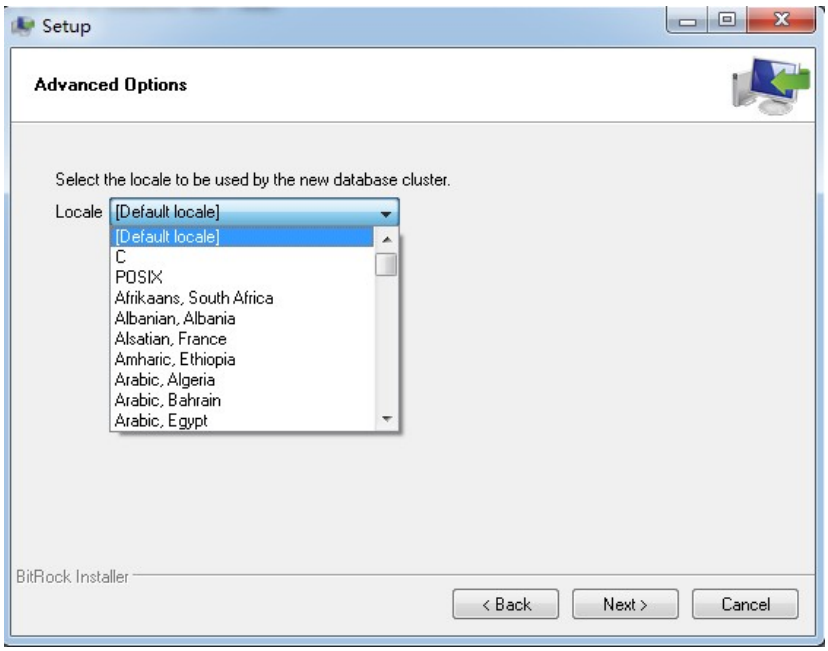


注：数据库超级用户是一个非管理员账户，这是为了减少黑客利用在 PostgreSQL 发现的缺陷对系统造成损害，因此需要对数据库超级用户设置密码，如下图所示，安装程序自动建立的服务用户的用户名默认为 postgres。

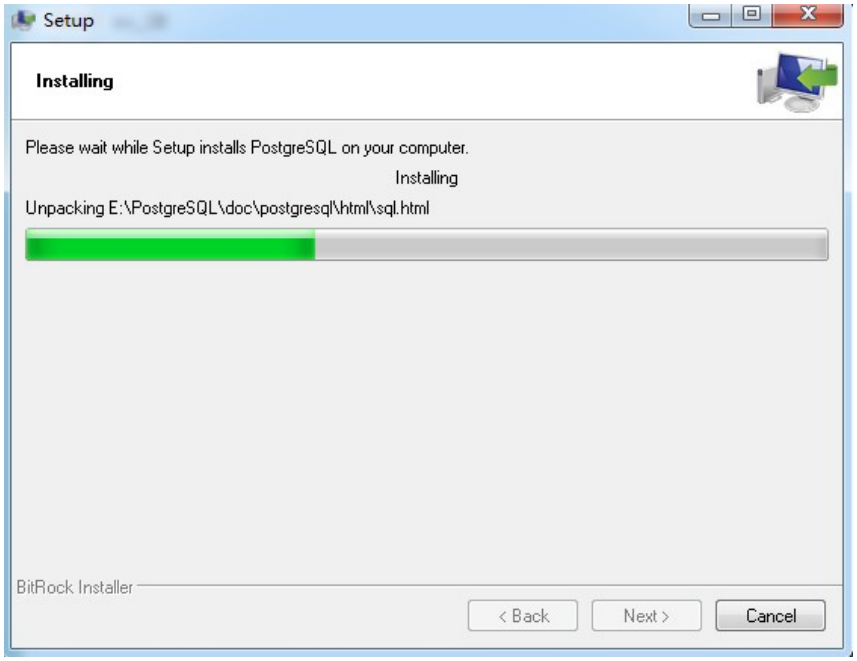
5、设置服务监听端口，默认为 5432



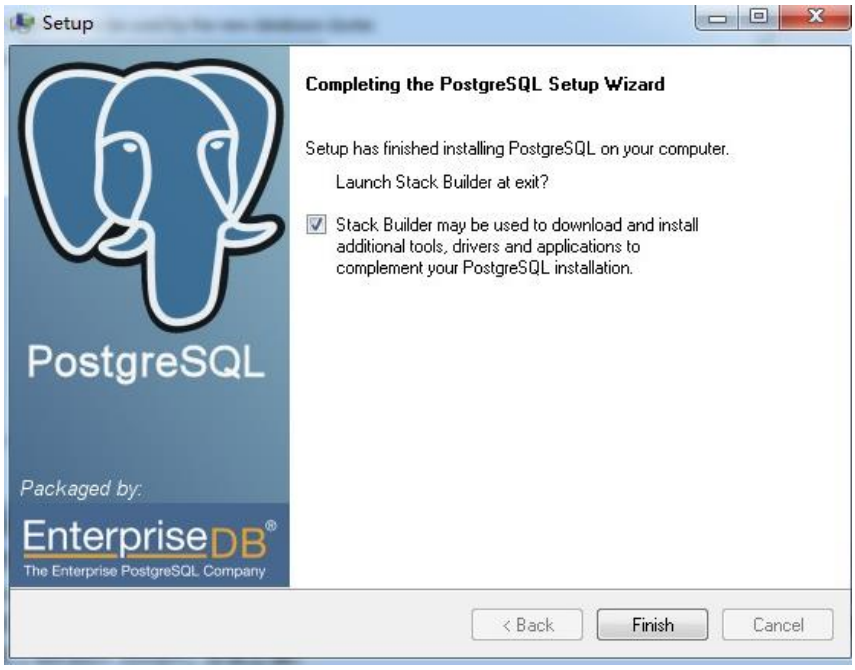
6、选择运行时语言环境



注：选择数据库存储区域的运行时语言环境（字符编码格式）。
在选择语言环境时，若选择"default locale"可能会导致安装不正确；同时，PostgreSQL 不支持 GBK 和 GB18030 作为字符集，如果选择其它四个中文字符集：中文繁体 香港（Chinese[Traditional], Hong Kong S.A.R.）、中文简体 新加坡（Chinese[Simplified], Singapore）、中文繁体 台湾（Chinese[Traditional], Taiwan）和中文繁体 澳门（Chinese[Traditional], Marco S.A.R.），会导致查询结果和排序效果不正确。建议选择"C"，即不使用区域。
----我选择了 default localt，安装正确；建议选择 default localt。
7、安装过程（2 分钟）



8、安装完成



注：多选框是安装额外的驱动和工具，可以不点。

在安装目录可以看到

名称	修改日期	类型	大小
bin	2013/2/25 11:03	文件夹	
data	2013/2/25 11:06	文件夹	
doc	2013/2/25 11:02	文件夹	
include	2013/2/25 11:02	文件夹	
installer	2013/2/25 11:03	文件夹	
lib	2013/2/25 11:03	文件夹	
pgAdmin III	2013/2/25 11:02	文件夹	
scripts	2013/2/25 11:02	文件夹	
share	2013/2/25 11:03	文件夹	
StackBuilder	2013/2/25 11:02	文件夹	
symbols	2013/2/25 11:03	文件夹	
pg_env.bat	2013/2/25 11:06	Windows 批处理...	1 KB
uninstall-postgresql.exe	2013/2/25 11:06	应用程序	6,073 KB

其中：data 存放数据文件、日志文件、控制文件、配置文件等。
uninstall-postgresql.exe 用于卸载已安装的数据库管理系统。
pg_env.bat 里配置了数据库的几个环境变量.

三、图形化界面 pgAdmin（大象）

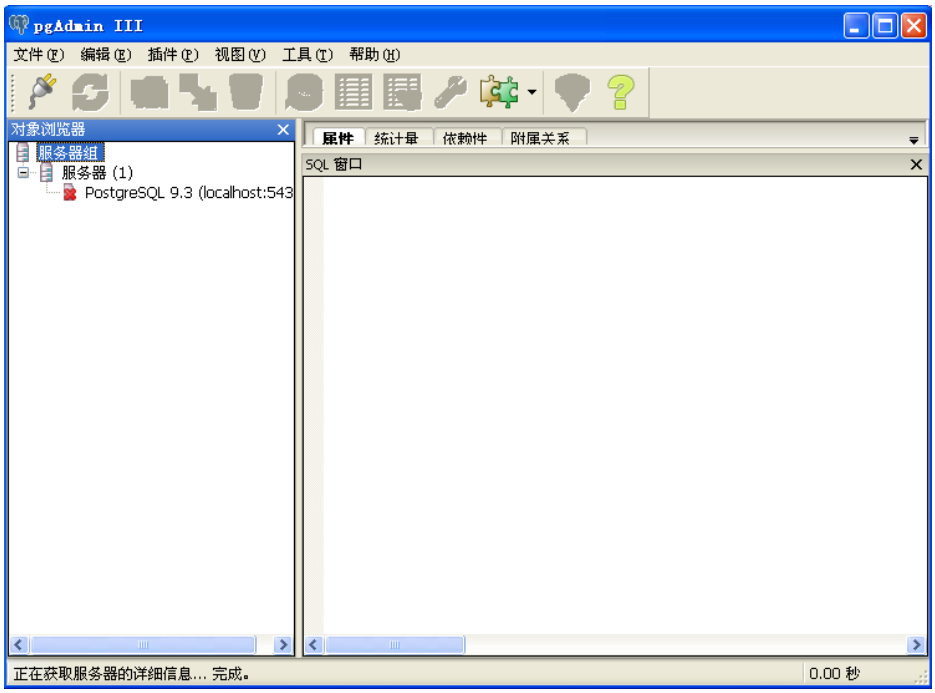
对于每种数据库管理系统，都有相当多的设计与管理工具（可视化界面管理工具），有的是数据库厂商自己提供的（一般都至少有一个），有的是第三方公司开发的，你甚至可以自己写一个简单易用的管理工具。例如 Oracle 的 Oracle SQL Developer（自己开发的）、PLSQL Developer（第三方公司开发的）、SQL Server Management Studio（自己开发的）、http://www.oschina.net/project（开源中国）网站上提供的个人或组织开发的简易小巧的管理工具。

PostgreSQL 就有好几款流行的管理工具，例如：pgAdmin、navicat_pgsql、phppgsql 等。

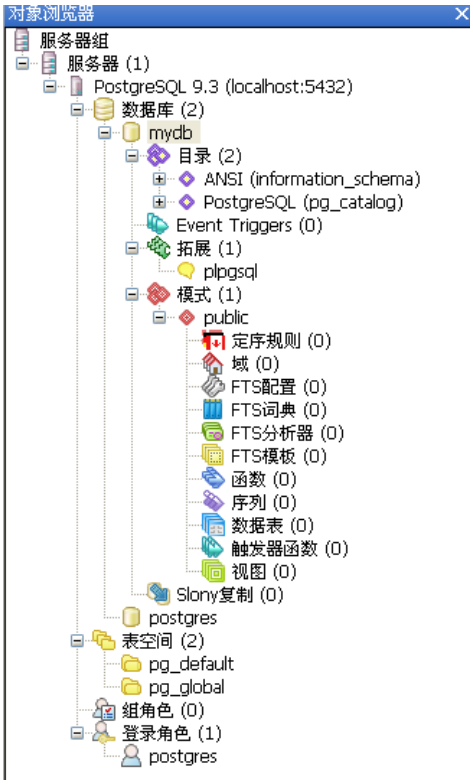
pgAdmin 是一个针对 PostgreSQL 数据库的设计和管理接口，可以在大多数操作系统上运行。软件用 C++编写，具有很优秀的性能。

pgadmin 是与 Postgres 分开发布的，可以从 www.pgadmin.org 下载。目前装个全功能的 PostgreSQL 数据库，自带该管理工具。

打开 pgAdmin，可以看到在第一部分安装的本地数据库的属性，如下图所示：



成功连接服务器后，如下图所示：

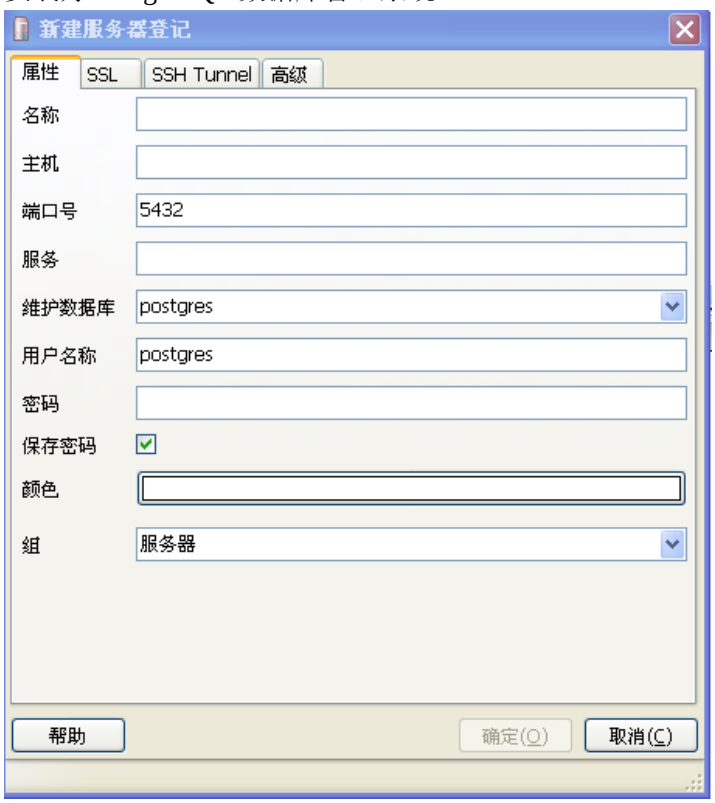


图中可以看出，新安装的 PostgreSQL 数据库管理系统带有一个数据库 postgres；已建好两个表空间：pg_default、pg_global。

initdb.exe 初始化的两个默认表空间 pg_global、pg_default。数据库默认的表空间 pg_default 是用来存储系统目录对象、用户表、用户表 index、和临时表、临时表 index、内部临时表的默认空间，他是模板数据库 template0 和 template1 的默认表空间。initdb.exe 初始化的两个默认表空间 pg_global、pg_default。数据库默认的表空间 pg_global 是用来存储共享系统目录的默认空间。

pg_default 为 PostgreSQL 也可以理解成系统表空间，它对应的物理位置为 $PGDATA/base 目录。
在 PostgreSQL（pg_catalog）下可以看到 postgers 数据库的一些数据字典和数据字典视图。

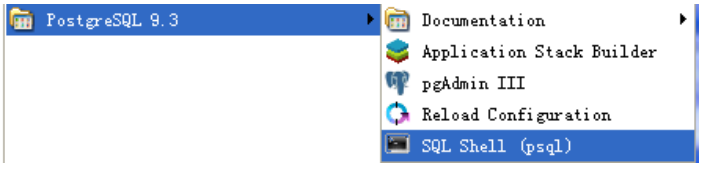
新建一个服务器连接，连接远程服务器上的 PostgreSQL 数据库（假设已有远程服务器上已安装好 PostgreSQL 数据库管理系统）：



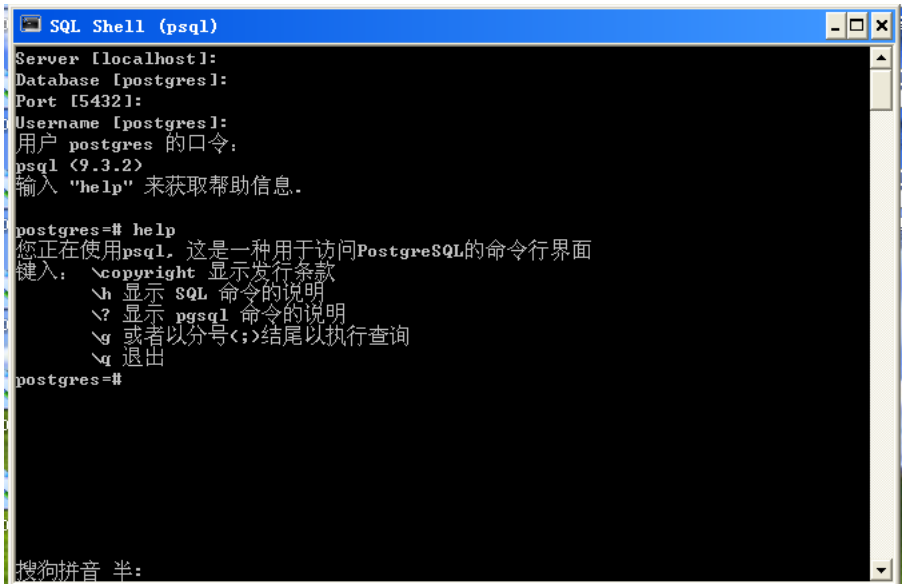
四、Pgsql

对于每种数据库管理系统，都会提供一个命令行管理接口，例如 Oracle 的 sqlplus，SQL Server 的 isql 和 osql 等。
凡是用图形管理界面可以实现的功能原则上都可以通过命令行界面命令实现。两者各有优缺点，使用场合不同。在 windows 下当然常用图形管理界面，因为在图像管理界面中往往都嵌有命令行工具，而在 unix 和 linux 下，当然就常用命令行工具了，除了我们在类 unix 下主要使用字符界面的原因外，还因为大部分情况下我们只能通过 telnet 或 ssh 工具远程连接服务器进行操作，此时也只能使用命令行了。

从开始目录打开 SQL shell（pgsql）



输入密码得到如下图界面：



五、PostgreSQL 常用数据类型

一、数值类型

名字	存储空间	描述	范围
smallint	2 字节	小范围整数	-32768 到 +32767
integer	4 字节	常用的整数	-2147483648 到 +2147483647
bigint	8 字节	大范围的整数	-9223372036854775808 到 9223372036854775807
decimal	变长	用户声明精度，精确	无限制
numeric	变长	用户声明精度，精确	无限制
real	4 字节	变精度，不精确	6 位十进制数字精度
double	8 字节	变精度，不精确	15 位十进制数字精度
serial	4 字节	自增整数	1 到 +2147483647
bigserial	8 字节	大范围的自增整数	1 到 9223372036854775807

1. 整数类型： 类型 smallint、integer 和 bigint 存储各种范围的全部是数字的数，也就是没有小数部分的数字。试图存储超出范围以外的数值将导致一个错误。常用的类型是 integer，因为它提供了在范围、存储空间和性能之间的最佳平衡。一般只有在磁盘空间紧张的时候才使用 smallint。而只有在 integer 的范围不够的时候才使用 bigint，因为前者(integer)绝对快得多。
2. 任意精度数值： 类型 numeric 可以存储最多 1000 位精度的数字并且准确地进行计算。因此非常适合用于货币金额和其它要求计算准确的数量。不过，numeric 类型上的算术运算比整数类型或者浮点数类型要慢很多。 numeric 字段的最大精度和最大比例都是可以配置的。要声明一个类型为 numeric 的字段，你可以用下面的语法： NUMERIC(precision,scale) 比如数字 23.5141 的精度为 6，而刻度为 4。 在目前的 PostgreSQL 版本中，decimal 和 numeric 是等效的。
3. 浮点数类型： 数据类型 real 和 double 是不准确的、牺牲精度的数字类型。不准确意味着一些数值不能准确地转换成内部格式并且是以近似的形式存储的，因此存储后再把数据打印出来可能显示一些缺失。

4. Serial(序号)类型： serial 和 bigserial 类型不是真正的类型，只是为在表中设置唯一标识做的概念上的便利。

```
CREATE TABLE tablename (
        colname SERIAL
 );
```

等价于

```
CREATE SEQUENCE tablename_colname_seq;
CREATE TABLE tablename(
        colname integer DEFAULT nextval('tablename_colname_seq') NOT NULL
);
```

这样，我们就创建了一个整数字段并且把它的缺省数值安排为从一个序列发生器取值。应用了一个 NOT NULL 约束以确保空值不会被插入。在大多数情况下你可能还希望附加一个 UNIQUE 或者 PRIMARY KEY 约束避免意外地插入重复的数值，但这个不是自动发生的。因此，如果你希望一个序列字段有一个唯一约束或者一个主键，那么你现在必须声明，就像其它数据类型一样。 还需要另外说明的是，一个 serial 类型创建的序列在其所属字段被删除时，该序列也将被自动删除，但是其它情况下是不会被删除的。因此，如果你想用同一个序列发生器同时给几个字段提供数据，那么就应该以独立对象的方式创建该序列发生器。

二、字符类型

名字	描述
varchar(n)	变长，有长度限制
char(n)	定长,不足补空白
text	变长，无长度限制

SQL 定义了两种基本的字符类型，varchar(n)和 char(n)，这里的 n 是一个正整数。两种类型都可以存储最多 n 个字符长的字串，试图存储更长的字串到这些类型的字段里会产生一个错误，除非超出长度的字符都是空白，这种情况下该字串将被截断为最大长度。

如果没有长度声明，char 等于 char(1)，而 varchar 则可以接受任何长度的字串。

这里需要注意的是，如果是将数值转换成 char(n)或者 varchar(n)，那么超长的数值将被截断成 n 个字符，而不会抛出错误。

最后需要提示的是，这三种类型之间没有性能差别，只不过是在使用 char 类型时增加了存储尺寸。虽然在某些其它的数据库系统里，char(n)有一定的性能优势，但在 PostgreSQL 里没有。在大多数情况下，应该使用 text 或者 varchar。

三、日期/时间类型

名字	存储空间	描述	最低值	最高值	分辨率
timestamp[无时区]	8 字节	包括日期和时间	4713 BC	5874897AD	1 毫秒/14 位
timestamp[含时区]	8 字节	日期和时间，带时区	4713 BC	5874897AD	1 毫秒/14 位
interval	12 字节	时间间隔	-178000000 年	178000000 年	1 毫秒/14 位
date	4 字节	只用于日期	4713 BC	32767AD	1 天
time[无时区]	8 字节	只用于一日内时间	00:00:00	24:00:00	1 毫秒/14 位

1. 日期/时间输入：任何日期或者时间的文本输入均需要由单引号包围，就象一个文本字符串一样。

日期格式

例子	描述
January 8, 1999	在任何 datestyle 输入模式下都无歧义
1999-01-08	ISO-8601 格式，任何方式下都是 1999 年 1 月 8 号，(建议格式)
1/8/1999	歧义，在 MDY 下是 1 月 8 号；在 DMY 模式下读做 8 月 1 日
1/18/1999	在 MDY 模式下读做 1 月 18 日，其它模式下被拒绝
01/02/03	MDY 模式下的 2003 年 1 月 2 日；DMY 模式下的 2003 年 2 月 1 日；YMD 模式下的 2001 年 2 月 3 日
1999-Jan-08	任何模式下都是 1 月 8 日
Jan-08-1999	任何模式下都是 1 月 8 日
08-Jan-1999	任何模式下都是 1 月 8 日
99-Jan-08	在 YMD 模式下是 1 月 8 日，否则错误
08-Jan-99	1 月 8 日，除了在 YMD 模式下是错误的之外
Jan-08-99	1 月 8 日，除了在 YMD 模式下是错误的之外
19990108	ISO-8601；任何模式下都是 1999 年 1 月 8 日
990108	ISO-8601；任何模式下都是 1999 年 1 月 8 日

时间格式

例子	描述
04:05:06.789	ISO 8601
04:05:06	ISO 8601
04:05	ISO 8601
040506	ISO 8601
04:05 AM	与 04:05 一样；AM 不影响数值
04:05 PM	与 16:05 一样；输入小时数必须 <= 12
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601

时间戳类型的有效输入由一个日期和时间的联接组成，后面跟着一个可选的时区。因此，1999-01-08 04:05:06 和 1999-01-08 04:05:06 -8:00 都是有效的数值。

2，示例

在插入数据之前先查看 datestyle 系统变量的值：

Sql: `show datestyle;`

输出窗口

	DateStyle text
1	ISO, YMD

```
INSERT INTO testtable(id,date_col) VALUES(1, DATE'01/02/03');  --datestyle 为YMD
```

```
 id   |  date_col
------+------------
  1   | 2001-02-03
```

```
set datestyle = MDY;
```

```
INSERT INTO testtable(id,date_col) VALUES(2, DATE'01/02/03');  --datestyle 为MDY
```

```
  id   |  date_col
-------+------------
   1   | 2001-02-03
   2   | 2003-01-02
```

```
INSERT INTO testtable(id,time_col) VALUES(3, TIME'10:20:00');  --插入时间。
```

```
  id   | time_col
-------+----------
   3   | 10:20:00
```

```
INSERT INTO testtable(id,timestamp_col) VALUES(4, DATE'01/02/03');
INSERT INTO testtable(id,timestamp_col) VALUES(5, TIMESTAMP'01/02/03 10:20:00');
```

```
  id   |    timestamp_col
-------+---------------------
   4   | 2003-01-02 00:00:00
   5   | 2003-01-02 10:20:00
```

五、布尔类型：

PostgreSQL 支持标准的 SQL boolean 数据类型。boolean 只能有两个状态之一：真(True)或 假(False)。该类型占用 1 个字节。

"真"值的有效文本值是：TRUE ， 't' ， 'true'， 'y' ， 'yes' ， '1'

而对于"假"而言，你可以使用下面这些：FALSE ， 'f' ， 'false' ， 'n' ， 'no' ， '0'

```
INSERT INTO testtable VALUES(TRUE, 'sic est');

INSERT INTO testtable VALUES(FALSE, 'non est');

 a |    b
---+---------
 t | sic est
 f | non est

SELECT * FROM testtable WHERE a;

 a |    b
---+---------
 t | sic est
```

六、位串类型：

位串就是一串 1 和 0 的字串。它们可以用于存储和视觉化位掩码。我们有两种类型的 SQL 位类型：bit(n)和 bit varying(n); 这里的 n 是一个正整数。bit 类型的数据必须准确匹配长度 n; 试图存储短些或者长一些的数据都是错误的。类型 bit varying 数据是最长 n 的变长类型；更长的串会被拒绝。写一个没有长度的 bit 等效于 bit(1)，没有长度的 bit varying 相当于没有长度限制。 针对该类型，最后需要提醒的是，如果我们明确地把一个位串值转换成 bit(n)，那么它的右边将被截断或者在右边补齐零，直到刚好 n 位，而不会抛出任何错误。类似地，如果我们明确地把一个位串数值转换成 bit varying(n)，如果它超过 n 位，那么它的右边将被截断。 见如下具体使用方式：

```
CREATE TABLE testtable (a bit(3), b bit varying(5));

INSERT INTO testtable VALUES (B'101', B'00');

INSERT INTO testtable VALUES (B'10', B'101');
```

这句 sql 执行时就会报错,因为.a 的数据的长度是 3,如果所插数据长度不为三则报错.

```
ERROR:  bit string length 2 does not match type bit(3)

INSERT INTO testtable VALUES (B'10'::bit(3), B'101');

  -----+-----
   101 | 00
   100 | 101

SELECT B'11'::bit(3);
```

```
 bit
-----
 110
(1 row)
```

注:(::)为转换符

七、数组类型：

1. 数组类型声明：

创建字段含有数组类型的表。

```
CREATE TABLE sal_emp (
    name            text,
    pay_by_quarter  integer[] --还可以定义为integer[4]或integer ARRAY[4]
);
```

插入数组数据

```
INSERT INTO sal_emp VALUES ('Bill', '{11000, 12000, 13000, 14000}');

INSERT INTO sal_emp VALUES ('Carol', ARRAY[21000, 22000, 23000, 24000]);
```

```
 name  |      pay_by_quarter
-------+---------------------------
 Bill  | {11000,12000,13000,14000}
 Carol | {21000,22000,23000,24000}
```

2 查询数组数据

和其他语言一样，PostgreSQL 中数组也是通过下标数字(写在方括弧内)的方式进行访问，只是 PostgreSQL 中数组元素的下标是从 1 开始 n 结束。

```
SELECT pay_by_quarter[3] FROM sal_emp;
```

```
 pay_by_quarter
----------------
          13000
          23000
```

```
SELECT name FROM sal_emp WHERE pay_by_quarter[1] <> pay_by_quarter[2];
```

PostgreSQL 中还提供了访问数组范围的功能，即 ARRAY[脚标下界:脚标上界]。

```
SELECT name,pay_by_quarter[1:3] FROM sal_emp;
```

```
--------+---------------------
 Bill   | {11000,12000,13000}
 Carol  | {21000,22000,23000}
```

3. 修改数组数据

代替全部数组值：

```
UPDATE sal_emp SET pay_by_quarter = '{31000,32000,33000,34000}' WHERE name = 'Carol';

--UPDATE sal_emp SET pay_by_quarter = ARRAY[25000,25000,27000,27000] WHERE name = 'Carol';  也可以。
```

```
 name  |      pay_by_quarter
--------+---------------------------
 Bill   | {11000,12000,13000,14000}
 Carol  | {31000,32000,33000,34000}
```

更新数组中某一元素：

```
UPDATE sal_emp SET pay_by_quarter[4] = 15000 WHERE name = 'Bill';
```

```
 name  |      pay_by_quarter
--------+---------------------------
 Carol  | {31000,32000,33000,34000}
 Bill   | {11000,12000,13000,15000}
```

更新数组某一范围的元素：

```
UPDATE sal_emp SET pay_by_quarter[1:2] = '{37000,37000}' WHERE name = 'Carol';
```

```
 name  |      pay_by_quarter
--------+---------------------------
 Bill   | {11000,12000,13000,15000}
 Carol  | {37000,37000,33000,34000}
```

直接赋值扩大数组：

```
UPDATE sal_emp SET pay_by_quarter[5] = 45000 WHERE name = 'Bill';
```

```
 name  |        pay_by_quarter
--------+-------------------------------
 Carol  | {37000,37000,33000,34000}
 Bill   | {11000,12000,13000,15000,45000}
```

八、复合类型

PostgreSQL 中复合类型有些类似于 C 语言中的结构体，也可以被视为 Oracle 中的记录类型，但是还是感觉复合类型这个命名比较贴切。它实际上只是一个字段名和它们的数据类型的列表。PostgreSQL 允许像简单数据类型那样使用复合类型。比如，表字段可以声明为一个复合类型。

```
CREATE TYPE inventory_item AS (
    name text,
```

```
    supplier_id integer,
    price numeric
);
```

和声明一个数据表相比，声明类型时需要加 AS 关键字，同时在声明 TYPE 时不能定义任何约束。下面我们看一下如何在表中指定复合类型的字段，如：

```
CREATE TABLE on_hand (
     item inventory_item,
    count integer
);
```

最后需要指出的是，在创建表的时候，PostgreSQL 也会自动创建一个与该表对应的复合类型，名字与表字相同，即表示该表的复合类型。

复合类型值输入：

我们可以使用文本常量的方式表示复合类型值，即在圆括号里包围字段值并且用逗号分隔它们。你也可以将任何字段值用双引号括起，如果值本身包含逗号或者圆括号，那么就用双引号括起，对于上面的 inventory_item 复合类型的输入如下：

```
'("fuzzy dice",42,1.99)'
```

如果希望类型中的某个字段为 NULL，只需在其对应的位置不予输入即可，如下面的输入中 price 字段的值为 NULL，

```
'("fuzzy dice",42,)'
```

如果只是需要一个空字串，而非 NULL，写一对双引号，如：

```
'("",42,)'
```

在更多的场合中 PostgreSQL 推荐使用 ROW 表达式来构建复合类型值，使用该种方式相对简单，无需考虑更多标识字符问题，如：

```
ROW('fuzzy dice', 42, 1.99)
ROW('', 42, NULL)
```

注：对于 ROW 表达式，如果里面的字段数量超过 1 个，那么关键字 ROW 就可以省略，因此以上形式可以简化为：

```
('fuzzy dice', 42, 1.99)
('', 42, NULL)
```

访问复合类型：

访问复合类型中的字段和访问数据表中的字段在形式上极为相似，只是为了对二者加以区分，PostgreSQL 设定在访问复合类型中的字段时，类型部分需要用圆括号括起，以避免混淆，如：

```
SELECT (item).name FROM on_hand WHERE (item).price > 9.99;
```

如果在查询中也需要用到表名，那么表名和类型名都需要被圆括号括起，如：

```
SELECT (on_hand.item).name FROM on_hand WHERE (on_hand.item).price > 9.99;
```

修改复合类型：

见如下几个示例：

```
--直接插入复合类型的数据，这里是通过 ROW 表达式来完成的。
INSERT INTO  on_hand(item)  VALUES(ROW("fuzzy dice",42,1.99));
--在更新操作中，也是可以通过 ROW 表达式来完成。
UPDATE on_hand  SET  item = ROW("fuzzy dice",42,1.99)  WHERE count = 0;
```

--在更新复合类型中的一个字段时，我们不能在 SET 后面出现的字段名周围加圆括号，但是在等号右边的表达式里引用同一个字段时却需要圆括号。

UPDATE on_hand SET item.price = (item).price + 1 WHERE count = 0;

--可以在插入中，直接插入复合类型中字段。

INSERT INTO on_hand (item.supplier_id, item.price) VALUES(100, 2.2);

九 postgreSQL jdbc 详解

当前 postgreSQL9.3 jdbc 有三种版本

Current Version

This is the current version of the driver. Unless you have unusual requirements (running old applications or JVMs), this is the driver you should be using. It supports Postgresql 7.2 or newer and requires a 1.5 or newer JVM. It contains support for SSL and the javax.sql package. It comes in two flavours, JDBC3 and JDBC4. If you are using the 1.6 then you should use the JDBC4 version. If you are using 1.7 or 1.8 then you should use the JDBC41 version.

JDBC3 Postgresql Driver, Version 9.3-1100

JDBC4 Postgresql Driver, Version 9.3-1100

JDBC41 Postgresql Driver, Version 9.3-1100

这是当前版本的驱动程序。除非你有不寻常的要求(运行旧的应用程序或 jvm),这是您应该使用的驱动程序。它支持 Postgresql 7.2 或更新,需要 1.5 或更新的 jdk。 它包含支持 SSL 和 javax.sql 包。 它有三种版本,JDBC3 JDBC4,jdbc41。 如果您使用的是 1.6,那么你应该使用 JDBC4 版本。如果您使用的是 1.7 或 1.8,那么你应该使用 JDBC41 版本。

十 postgreSQL 远程访问

安装 PostgreSQL 数据库之后，默认是只接受本地访问连接。如果想在其他主机上访问 PostgreSQL 数据库服务器，就需要进行相应的配置。配置远程连接 PostgreSQL 数据库的步骤很简单，只需要修改 data 目录下的 pg_hba.conf 和 postgresql.conf,其中 pg_hba.conf 是用来配置对数据库的访问权限，postgresql.conf 文件用来配置 PostgreSQL 数据库服务器的相应的参数。下面介绍配置的步骤：

1.修改 pg_hba.conf 文件，配置用户的访问权限：

```
77  # TYPE  DATABASE        USER            ADDRESS                 METHOD
78
79  # IPv4 local connections:
80  host    all             all             127.0.0.1/32            md5
81  host    all             all             192.168.85.12/24        md5
82
```

其中，第 81 行数据是新添加的内容，表示允许网段 192.168.85.12 上的主机使用所有合法的数据库用户名访问数据库，并提供加密的密码验证。(如果 IP 最后面的数字是 0,那么其中的数字 24 是子网掩码，表示允许 192.168.85.0--192.168.85.255 的计算机访问！)

2.修改 postgresql.conf 文件，将数据库服务器的监听模式修改为监听所有主机发出的连接请求。

定位到#listen_addresses=’localhost’。PostgreSQL 安装完成后，默认是只接受来在本机 localhost 的连接请求。将行开头都#去掉，将行内容修改为 listen_addresses=’*'来允许数据库服务器监听来自任何主机的连接请求