[PostgreSQL](http://blog.chinaunix.net/uid-15145533-id-2775821.html)

目录

# [9.0新特性介绍：唯一约束的增强](http://blog.chinaunix.net/uid-15145533-id-2775821.html)

2010-08-22

    在PostgreSQL9.0提供了一个新特性： Deferrable unique constraints, now permit mass updates to unique keys，基本意思是，对于可延迟的唯一约束，允许集中更新后再统一做约束检查。在PostgreSQL9.0以前的版本中，只有外键约束可以定义为deferrable，其它的都不行，而PostgreSQL9.0中，可为deferrable的约束类型还增加了UNIQUE, PRIMARY KEY, EXCLUDE。

    在PostgreSQL9.0版本之前，在执行DML语句时，唯一约束是不能deferable的，所以每更新一行，就要检查一次唯一性约束。所以，如果有一张表上有一个唯一字段叫id，内容是按顺序增长的，执行一条SQL语句： update t set id=id+1，这条SQL就会报错。实际上这条SQL语句如果更新完所有数据，是没有唯一性冲突的。这条SQL在oracle数据库中是可以正常执行完的，也不会报错。

    而PostgreSQL9.0之后，当创建的约束为“Deferable”后，不论你用set constraints all immediate，还是set constraints all deferrd时，这条SQL都不会了报错了，原因是当表上的唯一约束为“Deferable”后，当执行DML语句做批量更新时，是整个更新完后，才开始做唯一性检查，而不是每更新完一行后就检查。

见下面的实验：

环境为PostgreSQL9.0 beta4，

先不加deferable参数，约束默认是no deferable的：

postgres=# create table t4 (id int primary key,name varchar(40));

NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t4\_pkey" for table "t4"

CREATE TABLE

postgres=# insert into t4 values(1,'1111111111111111');

INSERT 0 1

postgres=# insert into t4 values(2,'2222222222222222');

INSERT 0 1

postgres=#

postgres=# update t4 set id=id+1;

ERROR:  duplicate key value violates unique constraint "t4\_pkey"

DETAIL:  Key (id)=(2) already exists.

STATEMENT:  update t4 set id=id+1;

ERROR:  duplicate key value violates unique constraint "t4\_pkey"

DETAIL:  Key (id)=(2) already exists.

默认约束是no deferable的，这时可以看到SQL执行出错。

执行set constraints all deferred:

postgres=# begin;

BEGIN

postgres=# set constraints all deferred;

SET CONSTRAINTS

postgres=# update t4 set id=id+1;

ERROR:  duplicate key value violates unique constraint "t4\_pkey"

DETAIL:  Key (id)=(2) already exists.

STATEMENT:  update t4 set id=id+1;

ERROR:  duplicate key value violates unique constraint "t4\_pkey"

DETAIL:  Key (id)=(2) already exists.

postgres=# end;

ROLLBACK

可以看出即使设置了constraints为deferred，由于建表时约束为no deferred的，也是就约束是不能延迟的检查，执行时，即使设置constraints为deferred时，也不生效。

下面我们把约束建成deferable的：

postgres=# drop table t4;

DROP TABLE

postgres=# create table t4 (id int primary key DEFERRABLE,name varchar(40));

NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t4\_pkey" for table "t4"

CREATE TABLE

postgres=# insert into t4 values(1,'1111111111111111');

INSERT 0 1

postgres=# insert into t4 values(2,'2222222222222222');

INSERT 0 1

postgres=# update t4 set id=id+1;

UPDATE 2

可以看到执行成功。

同样，我们在执行时，把constraints设置为immediate时，也可以执行成功：

postgres=# begin;

BEGIN

postgres=# set constraints all IMMEDIATE;

SET CONSTRAINTS

postgres=# update t4 set id=id+1;

UPDATE 2

postgres=# end;

COMMIT

如果执行时设置constraints为deferred时，然后执行一个会导致唯一约束冲突的更新，可以看到到提交时，才检查约束：

postgres=# begin;

BEGIN

postgres=# set constraints all deferred;

SET CONSTRAINTS

postgres=# update t4 set id=2 where id=1;

UPDATE 1

postgres=# end;

ERROR:  duplicate key value violates unique constraint "t4\_pk"

DETAIL:  Key (id)=(2) already exists.

STATEMENT:  end;

ERROR:  duplicate key value violates unique constraint "t4\_pk"

DETAIL:  Key (id)=(2) already exists.

# PostgreSQL之查找最慢的SQL 的方法

来自：http://blog.chinaunix.net/uid-24774106-id-3780341.html

[PostgreSQL](http://blog.csdn.net/shanzhizi)部署上之后，经过一段时间的运行，我们比较关心那些SQL运行时间比较长，或者说那些SQL执行的特别慢，拖累的性能，只有找到这些SQL，才能有针对性地对这些SQL进行优化，提升PostgreSQL的性能。  
   PostgreSQL提供了pg\_stat\_statements来存储SQL的运行次数，总运行时间，shared\_buffer命中次数，shared\_buffer read次数等统计信息。

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **References** | **Description** |
| userid | oid | [pg\_authid](http://www.postgresql.org/docs/9.1/static/catalog-pg-authid.html).oid | OID of user who executed the statement |
| dbid | oid | [pg\_database](http://www.postgresql.org/docs/9.1/static/catalog-pg-database.html).oid | OID of database in which the statement was executed |
| query | text |  | Text of the statement (up to [track\_activity\_query\_size](http://www.postgresql.org/docs/9.1/static/runtime-config-statistics.html" \l "GUC-TRACK-ACTIVITY-QUERY-SIZE) bytes) |
| calls | bigint |  | Number of times executed |
| total\_time | double precision |  | Total time spent in the statement, in seconds |
| rows | bigint |  | Total number of rows retrieved or affected by the statement |
| shared\_blks\_hit | bigint |  | Total number of shared blocks hits by the statement |
| shared\_blks\_read | bigint |  | Total number of shared blocks reads by the statement |
| shared\_blks\_written | bigint |  | Total number of shared blocks writes by the statement |
| local\_blks\_hit | bigint |  | Total number of local blocks hits by the statement |
| local\_blks\_read | bigint |  | Total number of local blocks reads by the statement |
| local\_blks\_written | bigint |  | Total number of local blocks writes by the statement |
| temp\_blks\_read | bigint |  | Total number of temp blocks reads by the statement |
| temp\_blks\_written | bigint |  | Total number of temp blocks writes by the statemen |

   上图表来自PostgreSQL官方文档，注意的一点是，我的PostgreSQL是9.1.9,此时total\_time的单位是秒，我观9.2的PostgreSQL的文档，total\_time单位已经是毫秒。所以我的参考文献[More On PostgreSQL perform](http://www.craigkerstiens.com/2013/01/10/more-on-postgres-performance/) 里面应该用的是9.2,因为外国这位大神默认单位是毫秒。  
   可以看出，pg\_stat\_statements统计了SQL的很多信息，方便我们分析SQL的性能。但是这个属于PostgreSQL的扩展，需要修改postgresql.conf，才能使用：  
操作步骤如下  
    1 修改配置文件，并且重启PostgreSQL方能生效

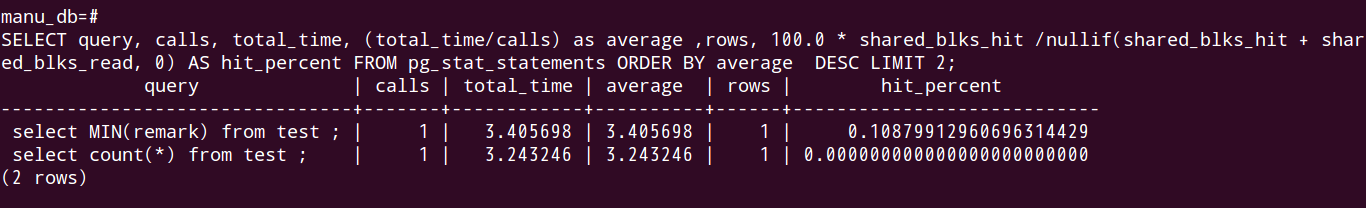
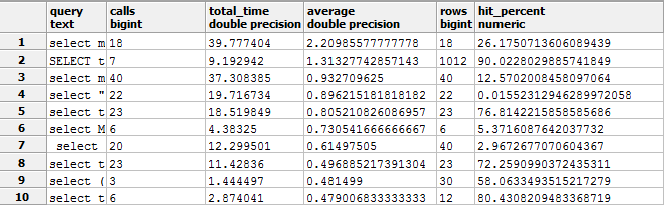
1. #------------------------------------------------------------------------------
2. # PG\_STAT\_STATEMENTS OPTIONS
3. #------------------------------------------------------------------------------
4. shared\_preload\_libraries = 'pg\_stat\_statements'
5. custom\_variable\_classes = 'pg\_stat\_statements'
6. pg\_stat\_statements.max = 1000
7. pg\_stat\_statements.track = all

    2 创建pg\_stat\_statements扩展

1. CREATE EXTENSION pg\_stat\_statements;

    从此之后，PostgreSQL就能记录SQL的统计信息。  
   上面的表格虽然丰富，其实我们基本比较关心执行最慢的SQL，如何查看执行最慢的10条SQL？

1. SELECT  query, calls, total\_time, (total\_time/calls) as average ,rows,
2. 100.0 \* shared\_blks\_hit /nullif(shared\_blks\_hit + shared\_blks\_read, 0) AS hit\_percent
3. FROM    pg\_stat\_statements
4. ORDER   BY average DESC LIMIT 10;

    我在我本地的DB，查找最慢的2条SQL，输出如下：  
     
   在我另一台机器上，用pgadmin查看：  
     
    统计结果一直都在，重启也不会清零，那么统计结果如何清零重新统计呢？执行下面SQL即可：

1. select pg\_stat\_statements\_reset() ;

    找到最耗时的SQL，我们就能针对这些耗时的SQL，查看是否有优化的余地。  
  
参考文献：  
1 [More on Postgres Performance](http://www.craigkerstiens.com/2013/01/10/more-on-postgres-performance/)2   PostgreSQL manual

# [**数据库备份策略**](https://www.cnblogs.com/hellojesson/p/8376030.html)

odoo默认的数据库为postgresql数据库，

PG是个非常强大的数据库，也是未来的一个趋势。

对于odoo的数据备份，odoo提供了自己的备份方式，

1. 从前台页面。输入odoo应用访问地址，例如：192.168.1.188:8069/web/database/manager 可以直接选择自己需要的数据库，进行手动复制，备份，和删除。

2. odoo应用社区也提供了一些第三方备份的模块，感兴趣的可以自行搜索。

3. 从数据库级别，即：按照常规数据库的备份恢复来操作。

    这里需要注意一点，对于单实例的非SAAS架构的odoo程序，完全可以按照数据库的常规方式去处理。

    但是，对于SAAS架构的odoo应用程序，就需要注意自己备份的数据库ID，恢复数据库的时候是否还能和主应用程序SAAS模块的管理ID相对应！

备份策略待更新！

# [**数据库 对比、同步工具推荐**](https://www.cnblogs.com/hellojesson/p/9205705.html)

1. dbForge Data Compare for PostgreSQL
2. PostgreSQL Data Sync

# [**Postgresql 字符串操作函数**](https://www.cnblogs.com/hellojesson/p/8044095.html)

样例测试：

update property set memorial\_no = btrim(memorial\_no, ' ') where memorial\_no like ' %'

或：update property set memorial\_no = trim(both ' ' from memorial\_no) where memorial\_no like ' %'

btrim()方法为删除字符串两边的某类字符，可以同时指定多个，在上面为' '，意为空格

trim()方法可以实现所有btrim()能实现的功能，事实上btrim()即为删除两边的某类字符串，trim()可以只指定某一边，当然也可以为两边

现把Postgresql的字符串操作函数罗列在以，以便日方使用

函数：string || string　  
说明：String concatenation 字符串连接操作  
例子：'Post' || 'greSQL'　＝　PostgreSQL

函数：string || non-string or non-string || string  
说明：String concatenation with one non-string input 字符串与非字符串类型进行连接操作  
例子：'Value: ' || 42 = Value: 42

函数：bit\_length(string)  
说明：Number of bits in string 计算字符串的位数  
例子：bit\_length('jose') = 32

函数：char\_length(string) or character\_length(string)  
说明：Number of characters in string 计算字符串中字符个数  
例子：char\_length('jose') = 4

函数：lower(string)  
说明：Convert string to lower case 转换字符串为小写  
例子：bit\_length('jose') = 32

函数：octet\_length(string)  
说明：Number of bytes in string 计算字符串的字节数  
例子：octet\_length('jose') = 4

函数：overlay(string placing string from int [for int])  
说明：Replace substring 替换字符串中任意长度的子字串为新字符串  
例子：overlay('Txxxxas' placing 'hom' from 2 for 4) = 4

函数：position(substring in string)  
说明：Location of specified substring 子串在一字符串中的位置  
例子：position('om' in 'Thomas') = 3

函数：substring(string [from int] [for int])  
说明：Extract substring 截取任意长度的子字符串  
例子：substring('Thomas' from 2 for 3) = hom

函数：substring(string from pattern)  
说明：Extract substring matching POSIX regular expression. See Section 9.7 for more information on pattern matching. 利用正则表达式对一字符串进行任意长度的字串的截取  
例子：substring('Thomas' from '...$') = mas

函数：substring(string from pattern for escape)  
说明：Extract substring matching SQL regular expression. See Section 9.7 for more information on pattern matching. 利于正则表达式对某类字符进行删除，以得到子字符串  
例子：trim(both 'x' from 'xTomxx') = Tom

函数：trim([leading | trailing | both] [characters] from string)   
说明：Remove the longest string containing only the characters (a space by default) from the start/end/both ends of the string 去除尽可能长开始，结束或者两边的某类字符，默认为去除空白字符，当然可以自己指定，可同时指定多个要删除的字符串  
例子：trim(both 'x' from 'xTomxx') = Tom

函数：upper(string)  
说明：Convert string to uppercase 将字符串转换为大写  
例子：upper('tom') = TOM

函数：ascii(string)  
说明：ASCII code of the first character of the argument. For UTF8 returns the Unicode code point of the character. For other multibyte encodings. the argument must be a strictly ASCII character. 得到某一个字符的Assii值  
例子：ascii('x') = 120

函数：btrim(string text [, characters text])  
说明：Remove the longest string consisting only of characters in characters (a space by default) from the start and end of string 去除字符串两边的所有指定的字符，可同时指定多个字符  
例子：btrim('xyxtrimyyx', 'xy') = trim

函数：chr(int)  
说明：Character with the given code. For UTF8 the argument is treated as a Unicode code point. For other multibyte encodings the argument must designate a strictly ASCII character. The NULL (0) character is not allowed because text data types cannot store such bytes. 得到某ACSII值对应的字符  
例子：chr(65) = A

函数：convert(string bytea, src\_encoding name, dest\_encoding name)   
说明：Convert string to dest\_encoding. The original encoding is specified by src\_encoding. The string must be valid in this encoding. Conversions can be defined by CREATE CONVERSION. Also there are some predefined conversions. See Table 9-7 for available conversions. 转换字符串编码，指定源编码与目标编码  
例子：convert('text\_in\_utf8', 'UTF8', 'LATIN1') = text\_in\_utf8 represented in ISO 8859-1 encoding

函数：convert\_from(string bytea, src\_encoding name)   
说明：Convert string to the database encoding. The original encoding is specified by src\_encoding. The string must be valid in this encoding. 转换字符串编码，自己要指定源编码,目标编码默认为数据库指定编码，  
例子：convert\_from('text\_in\_utf8', 'UTF8') = text\_in\_utf8 represented in the current database encoding

函数：convert\_to(string text, dest\_encoding name)   
说明：Convert string to dest\_encoding.转换字符串编码，源编码默认为数据库指定编码，自己要指定目标编码,  
例子：convert\_to('some text', 'UTF8') = some text represented in the UTF8 encoding

函数：decode(string text, type text)   
说明：Decode binary data from string previously encoded with encode. Parameter type is same as in encode. 对字符串按指定的类型进行解码  
例子：decode('MTIzAAE=', 'base64') = 123\000\001

函数：encode(data bytea, type text)   
说明：Encode binary data to different representation. Supported types are: base64, hex, escape. Escape merely outputs null bytes as \000 and doubles backslashes. 与decode相反，对字符串按指定类型进行编码  
例子：encode(E'123\\000\\001', 'base64') = MTIzAAE=

函数：initcap(string)  
说明：Convert the first letter of each word to uppercase and the rest to lowercase. Words are sequences of alphanumeric characters separated by non-alphanumeric characters. 将字符串所有的单词进行格式化，首字母大写，其它为小写  
例子：initcap('hi THOMAS') = Hi Thomas

函数：length(string)  
说明：Number of characters in string 讲算字符串长度  
例子：length('jose') = 4

函数：length(stringbytea, encoding name )  
说明：Number of characters in string in the given encoding. The string must be valid in this encoding. 计算字符串长度，指定字符串使用的编码  
例子：length('jose', 'UTF8') = 4

函数：lpad(string text, length int [, fill text])   
说明：Fill up the string to length length by prepending the characters fill (a space by default). If the string is already longer than length then it is truncated (on the right). 对字符串左边进行某类字符自动填充，即不足某一长度，则在左边自动补上指定的字符串，直至达到指定长度，可同时指定多个自动填充的字符  
例子：lpad('hi', 5, 'xy') = xyxhi

函数：ltrim(string text [, characters text])   
说明：Remove the longest string containing only characters from characters (a space by default) from the start of string 删除字符串左边某一些的字符，可以时指定多个要删除的字符  
例子：trim

函数：md5(string)  
说明：Calculates the MD5 hash of string, returning the result in hexadecimal 将字符串进行md5编码  
例子：md5('abc') = 900150983cd24fb0 d6963f7d28e17f72

函数：pg\_client\_encoding()  
说明：Current client encoding name 得到pg客户端编码  
例子：pg\_client\_encoding() = SQL\_ASCII

函数：quote\_ident(string text)  
说明：Return the given string suitably quoted to be used as an identifier in an SQL statement string. Quotes are added only if necessary (i.e., if the string contains non-identifier characters or would be case-folded). Embedded quotes are properly doubled. 对某一字符串加上两引号  
例子：quote\_ident('Foo bar') = "Foo bar"

函数：quote\_literal(string text)  
说明：Return the given string suitably quoted to be used as a string literal in an SQL statement string. Embedded single-quotes and backslashes are properly doubled. 对字符串里两边加上单引号，如果字符串里面出现sql编码的单个单引号，则会被表达成两个单引号  
例子：quote\_literal('O\'Reilly') = 'O''Reilly'

函数：quote\_literal(value anyelement)  
说明：Coerce the given value to text and then quote it as a literal. Embedded single-quotes and backslashes are properly doubled. 将一数值转换为字符串，并为其两边加上单引号，如果数值中间出现了单引号，也会被表示成两个单引号  
例子：quote\_literal(42.5) = '42.5'

函数：regexp\_matches(string text, pattern text [, flags text])  
说明：Return all captured substrings resulting from matching a POSIX regular expression against the string. See Section 9.7.3 for more information. 对字符串按正则表达式进行匹配，如果存在则会在结果数组中表示出来  
例子：regexp\_matches('foobarbequebaz', '(bar)(beque)') = {bar,beque}

函数：regexp\_replace(string text, pattern text, replacement text [, flags text])  
说明：Replace substring(s) matching a POSIX regular expression. See Section 9.7.3 for more information. 利用正则表达式对字符串进行替换  
例子：regexp\_replace('Thomas', '.[mN]a.', 'M') = ThM

函数：regexp\_split\_to\_array(string text, pattern text [, flags text ])  
说明：Split string using a POSIX regular expression as the delimiter. See Section 9.7.3 for more information. 利用正则表达式将字符串分割成数组  
例子：regexp\_split\_to\_array('hello world', E'\\s+') = {hello,world}

函数：regexp\_split\_to\_table(string text, pattern text [, flags text])  
说明：Split string using a POSIX regular expression as the delimiter. See Section 9.7.3 for more information. 利用正则表达式将字符串分割成表格  
例子：regexp\_split\_to\_table('hello world', E'\\s+') =   
hello  
world  
(2 rows)

函数：repeat(string text, number int)  
说明：Repeat string the specified number of times 重复字符串一指定次数  
例子：repeat('Pg', 4) = PgPgPgPg

函数：replace(string text, from text, to text)  
说明：Replace all occurrences in string of substring from with substring to 将字符的某一子串替换成另一子串  
例子：('abcdefabcdef', 'cd', 'XX') = abXXefabXXef

函数：rpad(string text, length int [, fill text])   
说明：Fill up the string to length length by appending the characters fill (a space by default). If the string is already longer than length then it is truncated. 对字符串进行填充，填充内容为指定的字符串  
例子：rpad('hi', 5, 'xy') = hixyx

函数：rtrim(string text [, characters text])  
说明：Remove the longest string containing only characters from characters (a space by default) from the end of string   
去除字符串右边指定的字符  
例子：rtrim('trimxxxx', 'x') = trim

函数：split\_part(string text, delimiter text, field int)  
说明：Split string on delimiter and return the given field (counting from one)  对字符串按指定子串进行分割，并返回指定的数值位置的值  
例子：split\_part([mailto:&amp;apos;abc~@~def~@~ghi&amp;apos;](mailto:%26amp;amp;apos;abc~@~def~@~ghi%26amp;amp;apos;), [mailto:&amp;apos;~@~&amp;apos;](mailto:%26amp;amp;apos;~@~%26amp;amp;apos;), 2) = def

函数：strpos(string, substring)  
说明：Location of specified substring (same as position(substring in string), but note the reversed argument order) 指定字符串在目标字符串的位置  
例子：strpos('high', 'ig') = 2

函数：substr(string, from [, count])  
说明：Extract substring (same as substring(string from from for count)) 截取子串  
例子：substr('alphabet', 3, 2) = ph

函数：to\_ascii(string text [, encoding text])  
说明：Convert string to ASCII from another encoding (only supports conversion from LATIN1, LATIN2, LATIN9, and WIN1250 encodings) 将字符串转换成ascii编码字符串  
例子：to\_ascii('Karel') = Karel

函数：to\_hex(number int or bigint)  
说明：Convert number to its equivalent hexadecimal representation 　对数值进行十六进制编码  
例子：to\_hex(2147483647) = 7fffffff

函数：translate(string text, from text, to text)   
说明：Any character in string that matches a character in the from set is replaced by the corresponding character in the to set 将字符串中某些匹配的字符替换成指定字符串，目标字符与源字符都可以同时指定多个  
例子：translate('12345', '14', 'ax') = a23x5

https://www.cnblogs.com/stephen-liu74/archive/2012/05/02/2294071.html

http://www.jsjtt.com/shujuku/postgresql/29.html

# [**postgresql重新启动、状态查看**](https://www.cnblogs.com/hellojesson/p/7149421.html)

场景描述：

当遇到数据库不能正常访问的时候，我们首先想到的是，查看相关的告警日志，一般先查看系统的日志，然后查看数据库的日志，Linux平台下，postgresql的日志文件存放目录在如下路径：

test@odoo:/var/log/postgresql$ pwd

/var/log/postgresql

将日志的信息实时打印输出：

test@odoo:/var/log/postgresql$ tail -f postgresql-9.5-main.log

如果需要重启数据库

利用psql启动**数据库**

**首先切换到postgres用户下，**

[postgres@test~]$ pg\_ctl start

查看系统中运行的postgres进程

#ps -ef | grep postgres

连接postgresql数据库

#psql -h 127.0.0.1 -d postgres -U postgres

停止postgresql数据库实例

#pg\_ctl stop

#ps -ef | grep postgres

启动服务器最简单的方法是像下面这样：

$ postgres -D /usr/local/pgsql/data

最好在后台启动postgres，使用下面的 Unix shell 语法：

$ postgres -D /usr/local/pgsql/data >logfile 2>&1 &

要做一次快速关闭：

$ kill -INT `head -1 /usr/local/pgsql/data/postmaster.pid`

如果启动失败提示一下信息：

LOG: could not bind IPv4 socket: Address already in use

HINT: Is another postmaster already running on port 5432? If not, wait a few seconds and retry.

FATAL: could not create TCP/IP listen socket

解决方法：

查看端口在用

netstat -lanp|grep 5432

ps -ef|grep postgres

498 673 1 0 10:50 ? 00:00:00/usr/bin/postgres -D /var/lib/cloudera-scm-server-db/data

kill进程  
kill -9 673

重启启动服务

service start

service status

 其他参考：

http://blog.csdn.net/Dracotianlong/article/details/7828515

# [**数据库默认端口**](https://www.cnblogs.com/hellojesson/p/6432149.html)

Oracle 默认端口：1521

Mysql 默认端口： 3306

PostgreSQL默认端口：5432

Redis默认端口：6379

Mogondb默认端口：27017

DB2 默认端口号为：5000

# PostgreSQL备份与恢复（全）

2017年09月07日 11:09:46  
**PostgreSQL**提供了三种**备份和恢复**的方式:SQL dump、文件系统复制和联机热备份。 每一种备份方式都有自己的优点和缺点，下面将详细介绍。  
  
9.1 SQL Dump  
     这种备份方式产生一个文本文件，里面包含创建各种数据库对象的SQL语句和每个表中的数据。另外，表上创建的索引中的数据不会被导出，只会导出索引的定义信息。在恢复数据库的时候，索引会被重建。可以使用数据库提供的工具pg\_dumpall和pg\_dump来进行备份。pg\_dumpall会备份一个数据库集群中的所有信息和数据。pg\_dump只备份数据库集群中的某个数据库的数据，它不会导出角色和表空间相关的信息，因为这些信息是整个数据库集群共用的，不属于某个单独的数据库。pg\_dump的基本用法如下：  
  
pg\_dump 数据库名 > 备份文件名  
  
pg\_dump将结果写到标准输出中，可以用操作系统的重定向命令将结果写到文件中。  
  
  
    可以在运行数据库的机器上执行pg\_dump命令，也可以在其它的机器上执行pg\_dump命令。 可以使用选项-h和-p来指定运行数据库的主机名和数据库监听的端口。例如：  
  
pg\_dump -h db\_server1 -p 5432  product > backup\_file   
  
该命令连接机器db\_server1上在端口5432处监听的数据库，将数据库product的数据备份到文件backup\_file中。如果pg\_dump命令没有使用-h和-p选项，将使用环境变量PGHOST的值作为机器名，使用环境变量PGPORT的值作为数据库的端口。如果用户没有定义环境变量PGHOST，默认使用本机名作为运行数据库的机器名。  
  
  
     默认的情况下，pg\_dump使用当前的操作系统用户作为连接数据库时使用的用户。可以使用选项-U或者设置环境变量PGUSER来指定连接数据库时使用的用户名。例如：  
  
pg\_dump -U liming -h db\_server1 -p 5432  product > backup\_file   
  
该命令使用用户liming连接机器db\_server1上在端口5432处监听的数据库，将数据库product的数据备份到文件backup\_file中。  
  
  
一般情况下，应该使用超级用户连接数据库进行备份操作，因为超级用户可以访问数据库中的任何信息。使用普通数据用户连接数据库，有些表可能无法访问。  
  
  
运行pg\_dump时，数据可以正常地执行其它操作。但ALTER TABLE这类修改数据库对象定义的操作会受到影响，可能会长时间处于等待状态而无法执行，所以在运行pg\_dump命令时，不要在数据库中运行修改数据库对象定义的操作。  
  
  
    另外要注意的是，如果数据库中有些表使用OID来实现外键约束，应当在备份数据库时同时备份表的OID信息，使用pg\_dump时加上选项–o即可达到这个目的。  
  
9.1.1 恢复数据库  
    pg\_dump创建的备份文件可以被工具psql识别。因此可以使用psql来读取pg\_dump创建的备份文件，实现恢复数据库的功能。例如：  
  
psql dbname < backup\_file  
  
psql后面的参数dbname指定的数据库必须已经存在。如果不存在，用户应当先创建dbname指定的数据库，然后再执行恢复数据的命令。psql也支持和pg\_dump一样的命令行选项，如-h和-p等。创建数据库dbname时，必须使用template0作为模板数据库，可以使用工具createdb创建数据库，也可以在psql中执行SQL命令create database来创建数据库。下面是两个实例：  
  
（1）createdb -T template0 dbname  
  
（2）create database dbname template=template0  
  
  
    另外，在执行恢复数据的操作以前，那些拥有数据库备份中的数据库对象或则对这些对象有访问权限的数据库的用户必须已经在数据库中存在，否则，恢复数据库以后，数据库备份中的数据库对象的所有者会发生改变。  
  
    默认的情况下，psql命令会一直执行下去直到结束，即使中间遇到SQL错误，恢复操作也会继续执行。如果想让psql在执行过程中遇到错误以后，停止恢复操作，可以在执行恢复操作以前，在psql中运行下面的命令：  
  
\set ON\_ERROR\_STOP  
  
  
    如果psql在执行过程中遇到错误，则只有一部分数据被正确地恢复，这时被恢复数据库中的数据是不完整的。psql提供了另外一种恢复模式，在这种模式下，一旦恢复操作执行过程中遇到任何错误，已经恢复的数据都会自动从数据库中被删除。可以使用psql的命令行选项-l或--single-transaction来打开这种模式。  
  
在恢复操作结束以后，应该使用ANALYZE命令来重新收集查询优化器统计数据。  
  
9.1.2 使用pg\_dumpall  
    pg\_dump只备份数据库集群中的某个数据库的数据，它不会导出角色和表空间相关的信息。pg\_dumpall则可以导出整个数据库集群中所有的数据库中的数据，同时也会导出角色、用户和表空间的定义信息。使用pg\_dumpall的一般命令格式如下：  
  
pg\_dumpall > backup\_file  
  
pg\_dumpall也支持和pg\_dump一样的命令行选项，如-h和-p等。同样可以使用psql来从pg\_dumpall创建的备份文件中恢复数据库。应该使用数据库超级用户来进行恢复数据库的操作。命令格式如下：  
  
psql -f backup\_file postgres  
  
  
     pg\_dumpall在执行的过程中，用postgres作为用户名来连接数据库。系统自动创建的数据库postgres中的内容也会被导出来，数据库template0和template1中的内容不会被导出来。  
  
9.1.3 大型数据库的备份和恢复  
     如果数据库的规模比较大，产生的备份文件的大小超级了操作系统能够允许的单个文件的大小的最大值，可以使用压缩和将备份文件分成对个部分这两个方法来解决这个问题。  
  
（1）采用压缩的方法，可以采用操作系统提供的任何一种压缩工具来实现，常用的是gzip。例如：  
  
pg\_dump dbname | gzip > filename.gz  
  
恢复时，使用下面的命令：  
  
gunzip -c filename.gz | psql dbname  
  
也可以使用下面的命令来恢复数据库：  
  
cat filename.gz | gunzip | psql dbname  
  
  
（2）将备份文件分成多个部分。使用操作系统的工具split来实现。例如:  
  
pg\_dump dbname | split -b 1m - filename  
  
在这个例子中，数据库备份被分成多个大小为1MB的文件。  
  
使用下面的命令进行恢复操作：  
  
cat filename\* | psql dbname  
  
  
（3）使用pg\_dump自带的压缩功能。这种方法产生的备份文件也是被压缩的，同第一种方法相比，它有一个优点，就是可以只恢复备份文件中的某个表的数据。这种方法的命令格式如下，就是增加了选项-Fc：  
  
pg\_dump -Fc dbname > filename  
  
不能使用psql命令恢复用这种方法备份的数据，必须使用pg\_restore来进行恢复操作。命令格式如下：  
  
pg\_restore -d dbname filename  
  
  
     对于非常大的数据库，可以将压缩与分割的方法同时使用（同时使用第一种和第二种方法，或者同时使用第二种和第三种方法）。  
  
9.2文件系统复制  
    文件系统复制这种方法是直接复制所有的数据库文件，存放到其它的存储介质上。这是最简单的备份数据库的方法。可以使用操作系统的命令来完成备份，例如：  
  
tar -cf backup.tar /usr/local/pgsql/data  
  
  
    复制数据文件以前，必须关闭数据库。这种备份方法产生的备份文件比较大，因为索引数据也会被备份。恢复数据库时只要把备份文件复制到存放数据文件的目录中即可。  
  
9.3 联机热备份与归档恢复  
9.3.1 联机热备份  
     进行联机热备份时，不用关闭数据库。数据库可以正常地执行其它操作。如果要使联机热备份，数据库必须运行在归档模式下，将参数数据库archive\_mode设为on，然后再将参数archive\_dir设成一个启动数据库的操作系统用户有读写权限的目录，数据库就会运行在归档模式。要使这两个参数生效，必须重新启动数据库。  
  
  
    进行联机热备份的步骤如下：  
  
（1）检查数据库是否运行在归档模式下。  
  
（2）用超级用户连接数据库（推荐使用psql），然后执行下面的命令：  
SELECT pg\_start\_backup('label');  
  
label是一个字符串，用来确定创建的备份，可以选取一个有明显的含义的名字作为label。  
  
pg\_start\_backup命令可能会执行比较长的时间才会结束，因为数据库会自动开始一个检查点操作。  
  
（3）使用操作系统命令(如cp)，将所有的数据库文件复制到其它的存储介质上。  
  
（4）执行下面的命令结束备份操作：  
  
SELECT pg\_stop\_backup();  
  
  
    备份操作结束以后，会在pg\_xlog子目录下产生一个备份描述文件，该文件以“.backup”结尾，例如000000010000000000000000.004535C0.backup。注意数据库归档进程会自动将备份操作产生的备份描述文件从pg\_xlog子目录复制到存放归档事务日志的目录中（参数archive\_dir指定的目录），如果在pg\_xlog目录中找不到备份描述文件，应该在存放归档事务日志的目录中去寻找它。恢复数据库的时候需要使用备份描述文件中的信息。备份描述文件中存放有下列信息：  
  
（1）开始事务日志文件名。  
  
（2）结束事务日志文件名。  
  
（3）检查点位置。  
  
（4）备份操作开始的时间。  
  
（5）备份操作结束的时间。  
  
（6）备份的名字（就是pg\_start\_backup命令中指定的名字）。  
  
  
    下面是一个备份描述文件的实例：  
  
START WAL LOCATION: 0/4535C0 (file 000000010000000000000000)  
  
STOP WAL LOCATION: 0/453A98 (file 000000010000000000000000)  
  
CHECKPOINT LOCATION: 0/4535C0  
  
START TIME: 2009-03-28 23:02:34 CST  
  
LABEL: b1  
  
STOP TIME: 2009-03-28 23:04:05 CST  
  
从该文件中可以看出备份操作开始的时间是2009-03-28 23:02:34，结束的时间是2009-03-28 23:04:05，备份的名字是b1，开始事务日志的名字是 000000010000000000000000，结束事务日志的名字也是 000000010000000000000000，检查点的位置是0/4535C0。  
  
从开始事务日志文件到结束事务日志文件之间的所有事务日志文件（包括这两个事务日志文件）必须被保存好，不能丢失，否则创建的数据库备份将是无效的，不能将数据库恢复到一个一致的状态。  
  
  
     备份操作在执行的过程中会在数据文件目下产生一个名为backup\_label的文件，该文件叫做备份标号文件。备份标号文件在备份操作结束以后会被系统自动删除。在执行上面的第三步操作的过程中，必须同时复制备份标号文件，因为恢复数据库的时候需要使用备份标号文件中的信息。  
  
  
9.3.2 归档恢复  
    进行归档恢复以前，应该准备好一个名为recovery.conf的文件，该文件中包含一些恢复操作的配置参数，这些参数决定恢复操作如何进行。下面详细介绍这些参数：  
  
  
（1）archive\_log\_dir  
  
该参数指定存放归事务日志的目录，所有需要的归档事务日志都应该存放在该目录中，系统在进行恢复操作时会自动从该目录中读取需要的事务日志文件。  
  
   
  
（2）recovery\_target\_time  
  
该参数指定一个时间，恢复操作进行到该时间时会自动停止。该参数用来实现时间点恢复（point-In-Time Recovery）。recovery\_target\_time和下面的recovery\_target\_xid只能指定一个。  
  
  
（3）recovery\_target\_xid  
  
该参数指定一个事务id，恢复操作进行到该事务时会自动停止。recovery\_target\_xid和上面的recovery\_target\_time只能指定一个。  
  
（4）recovery\_target\_inclusive  
  
该参数的值是true或false。默认值是true。它影响参数recovery\_target\_time和recovery\_target\_xid，如果它的值为true，恢复操作在指定的目标（时间或事务ID）以后停止，如果它的值为false，恢复操作在指定的目标以后停止。  
  
  
    参数archive\_log\_dir必须出现在recovery.conf的文件中，其它的参数则是可选的，如果recovery\_target\_xid和recovery\_target\_time都没有被指定，则默认恢复到最后一个事务日志文件确定的数据库的最近的状态。如果想进行时间点恢复，应该指定参数recovery\_target\_time。  
  
    下面是一个recovery.conf文件的实例，所有的参数的值都必须用两个单引号引起来：  
  
  
archive\_log\_dir = '/home/yan/archive\_log'  
  
recovery\_target\_time = '2004-07-14 22:39:00 EST'  
  
recovery\_target\_xid = '1100842'  
  
recovery\_target\_inclusive = 'true'          
  
  
    下面介绍进行归档恢复的具体步骤：  
  
（１）停止数据库服务器。将当前数据库备份到其它目录中。  
  
（２）准备好recovery.conf文件，将所有恢复操作需要的归档事务日志都存放在参数archive\_log\_dir指定的目录中，备份描述文件也必须被存放在参数archive\_log\_dir指定的目录中。  
  
（３）将以前创建的数据库备份复制到数据文件目录中（必须与以前的数据文件目录相同）。如果数据库使用了表空间，请验证pg\_tblspc子目录下面的每个符号链接是否有效。将准备好的recovery.conf文件存放到数据文件目录中。编辑文件pg\_hba.conf，不允许任何用户在恢复的过程中连接数据库。  
  
（４） 确保数据文件目录中存在一个名为backup\_label的文件。删除pg\_xlog子目录中的所有文件，重新在pg\_xlog中创建一个名为archive\_status的子目录。  
  
（５）启动数据库，数据库在启动以后将自动进行恢复操作，恢复操作成功完成以后，数据库将自动打开，进入正常的工作状态。恢复操作成功以后，系统会将文件recovery.conf重命名为recovery.done。  
  
（６）检查数据库中的内容是否正确。  
  
   
  
     归档恢复成功结束以后，数据库会自动打开，进入正常的工作状态，可以开始响应用户的连接请求，应该修改pg\_hba.conf文件，允许用户连接数据库。归档恢复成功结束以后，在数据库的运行日志中会有下面的提示信息：  
  
……  
  
日志： 00000： 归档恢复结束。  
  
      
  
    文件backup\_label在恢复操作执行结束以后会被自动重命名为backup\_label.old。确定归档恢复成功以后，应该删除backup\_label.old，因为它已经没有任何作用。  
  
9.3.3 注意事项  
    进行归档恢复时，有下面几个注意事项：  
  
（１）哈希索引上面的操作没有被记录到事务日志中，归档恢复完成以后，必须对每个哈希索引执行REINDEX操作。  
  
（２）在创建数据库备份时不要修改模板数据库。  
  
9.3.4 时间线（timeline）  
     时间线是[**PostgreSQL**](http://blog.csdn.net/shanzhizi/)独有的概念。它是一个整数值，与归档恢复有关。在用initdb创建一个初始的数据库集群以后，该数据库集群的时间线是１。每进行一次归档恢复，就会产生一个新的时间线，新的时间线的值在上一个时间线的值的基础上加一。每次归档恢复完成以后，都会产生一个时间线历史文件，该文件以“.history”结尾，例如00000002.history。时间线历史文件首先被存放在pg\_xlog目录中，数据库归档进程以后会自动将时间线历史文件从pg\_xlog子目录复制到存放归档事务日志的目录中（参数archive\_dir指定的目录）。