## PostgreSQL简介

相对于MySQL的弱点，postgreSQL有以下几个优点：

1. 功能强大：支持所有主流的多表连接查询的方式（如：“nest loop”，“hash join”，“sort merge join”等）；支持绝大多数的sql语法。
2. 性能优化工具与度量信息丰富：postgresql数据库中有大量的性能视图，可方便地定位问题
3. 在线功能好
4. 支持同步复制功能，通过master和slave之间的复制可实现零数据丢失的高可用方案。

## SQL语言入门

在SQL shell下：\c dbname; \l 查看数据库；

Psql –h <hostname or ip> -p <端口> [数据库名称] [用户名称]

\d [pattern] \dt; \d tablename; 索引：/di 序列：/ds 视图：/dv 函数：/df 列出所有的schema: \dn;

如果想显示SQL已执行的时间，可以使用\timing

“ORDER BY”子句放到“WHERE”之后，加DESC后倒序排列。

分组查询：”GROUP BY”,需要使用聚合函数，常用的有sum、count等。

TRUNCATE TABLE与不带where条件子句的belete语句从执行结果上看，两者实现了相同的功能，但两者实现的原理不一样。 TRUNCATE TABLE是DDL（数据定义）语句，相当于重新定义一个新表的方法把原先表的内容直接丢弃了；而delete是DML语句，把数据一条一条的删除，若要删除很多行数据，会比较慢。

## 五、数据类型

对于所有类型都可以使用“类型名”加上单引号括起来的类型值格式进行输入。

Select bit ‘1110011’; select int ‘1’ + int ‘2’;

类型转换函数CAST

Select Cast(‘5’ as int), Cast(‘2014-07-17’ as date);

更简洁的类型转换方式：双冒号方式

Select ‘5’::int, ‘2014-07-17’::date;

5.3数值类型

Smallint：2字节 -215~215-1 int或integer：4字节 -231~231-1； bigint:8

Numeric或decimal：变长 用户声明的精度，精确 real:4字节 变精度，不精确，6位十进制数字精度

Double precision：8字节 15位十进制数字精度 serial：4字节 ，自增整数。 Bigserial：8字节。

NUMERIC(precision,scale) 精度precision必须为正数，标度scale可以为零或正数

序列类型

CREATE TABLE t (id SERIAL);

等价于

CREATE SEQUENCE t\_id\_seq;

CREATE TABLE t ( id integer NOT NULL DEFAULT nextval(‘t\_id\_seq’) );

ALTER SEQUENCE t\_id\_seq OWNED BY t\_id;

5.4 字符串类型

Char\_length(string);字符个数 convert(string using conversion\_name)

Octet\_length(string);字节数

Overlay(string placing string from int [for int]);替换子字符串

Overlay(‘Txxxxas’ placing ‘hom’ from 2 for 4); Thomas

Position(substring in string) 指定的子字符串的位置 position（‘om’ in ‘Thomas’）; 3

Substring(‘Thomas’ from 2 for 3); hom

Trim([leading | trailing | both] [characters] from string); 从字符串string的开头/结尾/两边删除只包含characters中字符（默认是一个空白）最长的字符串

Trim(both ‘x’ from ‘xTxomxx’); Txom

Length(string); decode(string text, type text); encode(data bytea, type text);

Md5(string); replace(string text, from text, to text)

Rpad(string text,length int [, fill text]); rpad(‘os’, 7, ‘123’); os12312

Split\_part(string text, delimiter text, field int); 根据delimiter分隔string返回生成的的field个子字符串（1为基）；

Strops(string, substring); 指定的子字符串的位置，同position(substring in string)

Translate(string text, from text, to text); 把在string中包含的任何与from中字符匹配的字符转化为对应的在to中的字符。 Translate（‘12345’,’14’,’db’）; d23b5

5.5二进制数据类型

二进制字符串适用于存储那些程序员认为是“原始字节”的数据，比如图片内容，而字符串则适合存储文本。

5.7 日期/时间类型

Show datestyle; set datestyle=’YMD’; create table t(col1 date);

Insert into t values(date ‘2010-12-10’);

Age(timestamp, timestamp);减去参数后的“符号化”结果

Age(timestamp); 从current\_date减去参数后的结果； age(timestamp ‘1980-01-01’); 38 years 5 mons 3 days;

Current\_date; current\_time; current\_timestamp;

Date\_part(text, timestamp); 获取子域 date\_part(‘hour’, timestamp ‘2011-09-12 20:32:22’); 20

Extract(field from timestamp);同获取子域

Date\_trunc(text,timestamp);截取成指定的精度

Date\_trunc(‘hour’, timestamp ‘2001-02-16 20:38:40’) 2001-02-16 20:00:00

5.8枚举类型

Create type week as enum(‘sun’,’mon’,’tues’,’wed’,’thur’,’fri’,’sat’);

Enum\_first(anyenum); enum\_last(anyenum); enum\_range(anyenum,anyenum);

5.9网络地址类型

Cidr:IPv4或IPv6的网络地址。 Inet：IPv4或IPv6网络地址或主机地址 macaddr：以太网MAC地址。

5.11复合类型

复合类型常量的一般格式如下：‘（val1,val2,….）’

CREATE TYPE person AS (name text, age integer, sex Boolean);

CREATE TABLE author (id int, person\_info person, book text);

Insert into author values(1, ‘(“张三”, 29, TRUE)’, ‘张三的自传’);

也可以用ROW表达式语法来构造复合类型值

Insert into author values(4, ROW(‘张三’, 29, TRUE), ‘自传’);

访问复合类型： select (author.person\_info).name from author;

5.12XML类型

5.13JSON类型

5.14Range类型

此类型可以进行范围快速搜索。

CREATE TYPE inetrage AS RANGE (subtype = inet);

内置常用range类型：int4range,int8range,numrange,tsrange无时区的时间戳范围类型，tstzrange,daterange

SUBTYPE=subtype:指定子类型； SUBTYPE\_OPCLIASS=subtype\_operator\_class:指定子类型的操作符； SUBTYPE\_DIFF=subtype\_diff\_function:定义子类型的差别函数。

5.15数组类型

5.16伪类型

Any anyelement anyarray anynonarray……

## 六、逻辑结构管理

### 6.3模式

可以理解为一个命名空间或目录。不同的模式下可以有相同名称的表、函数等对象且互相不冲突。

### 6.4 表

Create table test01（id int primary key）;

Create table test01 (id1 int, id2 int, constraint pk\_test02 primary key(id1,id2));

Constraint constraint\_name unique(col1\_name, col2\_name,…)

Constraint constraint\_name check(expression)

Constraint constraint\_name references tablename(col\_name)

CREATE TABLE bady2 (LIKE child INCLUDING ALL);其中including all是把所有的属性都复制过来。

也可以使用：”CREATE TABLE … AS”来创建表

CREATE TABLE bady2 AS SELECT \* FROM child WITH NO DATA;

临时表

1.会话级的临时表 2.事务级的临时表

不能访问其他session的临时表 在默认情况下创建的临时表是会话级的，如果想创建出事务级的临时表，可以加“ON COMMIT DELETE ROWS”子句。

ON COMMIT PRESERVE ROWS ON COMMIT DELETE ROWS ON COMMIT DROP

修改表

增加字段 alter table class add column class\_teacher varchar(40) check (class\_teacher <> ‘’);

删除字段 alter table class drop column class\_no cascade;

在删除字段时如果这个字段被另一个表的外键所引用，删除时则会报错，如果想删除外键依赖，需要使用“CASCADE”

#### 6.4.7表继承及分区表

Create table persons (name text, age int, sex Boolean);

Create table students(class\_no int)inherits (persons);

当查询父表时，会把这个父表中子表的数据也查询出来，反之则不行。

如果只想把父表本身的数据查询出来，只需要在查询的表名前加“only”关键字

所有父表的检查约束和非空约束都会自动被所有子表继承。不过其他类型的约束（唯一、主键、外键）则不会被继承。

分区表 表分区就是把逻辑上的一个大表分割成物理上的几个小块。

建分区表的步骤：

1. 创建父表，所有分区都从它继承，这个表中没有数据。
2. 创建几个子表，每个表都是从主表继承而来。通常，这些表不会增加任何字段。
3. 给分区表增加约束，定义每个分区允许的键值
4. 对于每个分区，在关键字字段上创建一索引，也可创建其他你想创建的索引
5. 定义一个规则或者触发器，把对主表的数据插入重定向到合适的分区表。
6. 确保constraint\_exclusion里的配置参数postgresql.conf是打开的。打开后，如果查询中where子句的过滤条件与分区的约束条件匹配，那么这个查询会智能地只查询这个分区，而不会查询其他分区。

示例：

主表： CREATE TABLE sales\_detail ( product\_id int not null, price numeric(12,2), amount int not null, sale\_date date not null, buyer varchar(40), buyer\_contact text);

分区表：CREATE TABLE sales\_detail\_y2014m01 (CHECK (sale\_date >= DATE‘2014-01-01’AND sale\_date <DATE’2014-02-01’) ) INHERITS (sales\_detail);

CREATE TABLE sales\_detail\_y2014m01 (CHECK (sale\_date >= DATE ‘2014-02-01’ AND sale\_date <DATE ’2014-03-01’) ) INHERITS (sales\_detail);

CREATE TABLE sales\_detail\_y2014m01 (CHECK (sale\_date >= DATE ‘2014-03-01’ AND sale\_date <DATE ’2014-04-01’) ) INHERITS (sales\_detail);

……

在分区键“sale\_date”上建索引：

CREATE INDEX sale\_detail\_y2014m01\_sale\_date ON sales\_detail\_y2014m01 (sale\_date);

CREATE INDEX sale\_detail\_y2014m02\_sale\_date ON sales\_detail\_y2014m02 (sale\_date);

CREATE INDEX sale\_detail\_y2014m03\_sale\_date ON sales\_detail\_y2014m03 (sale\_date);

为了在往sale\_detail表中插入数据时，自动且正确地把数据插入到正确的分区，建立触发器：

CREATE OR REPLACE FUNCTION sale\_detail\_insert\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF ( NEW.sale\_date >= DATE ‘2014-01-01’ AND NEW.sale\_date < DATE ‘2014-02-01’) THEN

INSERT INTO sales\_detail\_y2014m01 values(NEW.\*);

ELSEIF ( NEW.sale\_date >= DATE ‘2014-02-01’ AND NEW.sale\_date < DATE ‘2014-03-01’) THEN

INSERT INTO sales\_detail\_y2014m01 values(NEW.\*);

ELSEIF ( NEW.sale\_date >= DATE ‘2014-03-01’ AND NEW.sale\_date < DATE ‘2014-04-01’) THEN

INSERT INTO sales\_detail\_y2014m01 values(NEW.\*);

ELSE

RAISE EXCEPTION ‘Date out of range. Fix the sale\_detail\_insert\_trigger() function!’;

END IF;

RETURN NULL;

END

$$

LANGUAGE plpgsql;

CREATE TRIGGER insert\_sale\_detail\_trigger BEFORE INSERT ON sale\_detail FOR EACH ROW EXECUTE PROCEDURE sale\_detail\_insert\_trigger();

以上是使用触发器把插入的数据重新定位到相应的分区中的，也可以使用“规则”来实现同样的功能。

CREATE RULE sales\_detail\_insert\_y2014m01 AS

ON INSERT TO sales\_detail WHERE

(sale\_date>=DATE’2014-01-01’ AND sale\_date <DATE’2014-02-01’)

DO INSTEAD

INSERT INTO sales\_detail\_y2014m01 VALUES (NEW.\*);

CREATE RULE sales\_detail\_insert\_y2014m02 AS

ON INSERT TO sales\_detail WHERE

(sale\_date>=DATE’2014-02-01’ AND sale\_date <DATE’2014-03-01’)

DO INSTEAD

INSERT INTO sales\_detail\_y2014m02 VALUES (NEW.\*);

……

### 6.5触发器

是一种由事件自动触发执行的特殊的存储过程，这些事件可以是对一个表进行INSERT/UPDATE/DELETE等操作。

触发器经常用于加强数据的完整性约束和业务规则上的约束等

建触发器的步骤：先为触发器建一个执行函数，此函数的返回类型为触发器类型；然后即可创建相应的触发器。

#### 6.5.2语句级触发器与行级触发器

CREATE FUNCTION log\_student\_trigger()

RETURNS trigger AS

$$

BEGIN

INSERT INTO log\_student values(now(), user, TG\_OP);

RETURN NULL;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER log\_student\_trigger

AFTER INSERT OR DELETE OR UPDATE ON student

FOR STATEMENT EXECUTE PROCEDURE log\_student\_trigger();

CREATE TRIGGER log\_student\_trigger2

AFTER INSERT OR DELETE OR UPDATE ON student

FOR ROW EXECUTE PROCEDURE log\_student\_trigger();

#### 6.5.5触发器的行为

触发器函数有返回值。语句级触发器应该总是返回NULL,即必须显式地在触发器函数中写上“RETURN NULL”，如果没有写，将导致出错。

#### 6.5.6触发器函数中的特殊变量

NEW\OLD\TG\_NAME\TG\_WHEN\TG\_LEVEL\TG\_OP\TG\_RELID\TG\_RELNAME\TG\_TABLE\_NAME\TG\_TABLE\_SCHEMA\TG\_NARGS\TG\_ARGV[]

6.6事件触发器

目前事件触发器支持以下三种DDL事件：

1. ddl\_command\_start:一个DDL开始执行前被触发
2. ddl\_command\_end:一个DDL执行完成后被触发
3. sql\_drop:删除一个数据库对象前被触发

一个禁止所有DDL语句的例子：

CREATE OR REPLACE FUNCTION abort\_any\_command()

RETURNS event\_trigger

LANGUAGE plpgsql

AS $$

BEGIN

RAISE EXCEPTION ‘command % is disabled’, tg\_tag;

END;

$$;

CREATE EVENT TRIGGER abort\_DDL ON DDL\_command\_start

EXECUTE PROCEDURE abort\_any\_command();

### 6.7表空间

需要把不同的表放到不同的存储介质或不同文件系统下时，需要使用表空间。表空间实际上是为表指定一个存储的目录

CREATE TABLESPACE tbs\_data location ‘/data/pgdata’;

CREATE database db01 tablespace tbs\_data;

Alter database db01 set tablespace tbs\_data;

Alter table test01 set tablespace pg\_default;

### 6.8视图

视图就是由查询语句定义的虚拟表。

复杂的查询易于理解和使用；安全；把一些函数返回的结果映射成视图

创建视图

Create view vw\_users AS SELECT id, user\_name, user\_email, user\_mark FROM users;

也可以加关键字TEMP或temporary来建临时视图；

可更新视图

可以通过定义一个规则来达到更新视图的目的

CREATE RULE vw\_users\_upd AS

ON UPDATE TO vw\_users DO instead update users set user\_email=new.user\_email;

从postgresql9.1开始，可通过“INSTEAD OF”的触发器更新视图

Create or replace function vw\_users\_insert\_trigger()

Returns trigger as $$

Begin

Insert into users values(new.id, new.user\_name, ‘11111’, new.user\_email,new.user\_mark);

Return null;

End

$$

Language plpgsql;

Create trigger insert\_vw\_users\_trigger

Instead of insert on vw\_users

For each row execute procedure vw\_user\_insert\_trigger();

### 6.9索引

好处是加快对表中记录的查找或排序，代价是增加了数据库的存储空间、在插入和修改数据时要花费较多时间

索引的分类：

B-tree:最常用的索引，适合处理等值查询和范围查询。

Hash：只能处理简单的等值查询。

GiST：不是一种单独的索引类型，而是一种架构，可以在这种架构上实现很多不同的索引策略。GiST索引定义的特定操作符可以用于特定索引策略。

SP-GiST:即空间分区GiST索引

GIN：反转索引，它可以处理包含多个键的值，如数组等

创建索引

CREATE TABLE contacts(id int primary key, name varchar(40), phone varchar(32)[], address text);

在字段“name”上建一个B-tree索引：

CREATE INDEX idx\_contacts\_name on contacts(name);

Phone字段是一个数组，b-tree不再起作用，这时可以建一个GIN索引：

CREATE INDEX idx\_contacts\_phone on contacts using gin(phone);

这时就可进行快速查询了，假设想查询“13422334455”是谁的

SELECT \* from contacts where phone @> array[‘13422334455’::varchar(32)];

还可以指定存储参数”WITH(storage\_parameter=value)”,常用的存储参数为FILLFACTOR

CREATE INDEX idx\_contacts\_name on contacts(name) WITH (FILLFACTOR=50);

按降序建索引：

CREATE INDEX idx\_contacts\_name on contacts(name desc);

CREATE INDEX idx\_contacts\_name on contacts(name DESC NULLS FIRST);

6.9.4并发创建索引

通常在创建索引的时候postgresql会锁定表以防止写入，然后对表做全表扫描，从而完成创建索引操作。可以读取，但是增、删、改将被阻塞到索引创建完毕。

加入CONCURRENTLY来实现并发创建索引。

CREATE INDEX CONCURRENTLY index\_texttab01\_note on testtab01(note);

修改索引

ALTER INDEX name RENAME TO new\_name;

ALTER INDEX name SET TABLESPACE tablespace\_name

### 6.10用户及权限管理

CREATE ROLE name [ [ with ] option [ …]] create user name [ [ with ] option [ … ] ]

ALTER ROLE name [ [ with ] option [ … ] ]

GRANT some\_privileges ON database\_object\_type object\_name TO role\_name;

6.11事务、并发、锁 ACID

#### 6.11.3事务的使用

\set AUTOMMIT off;

\echo :AUTOCOMMIT;

使用“BEGIN”语句开启一个事务。

SAVEPOINT： 把操作过程分成几个部分，若后面执行失败，则回滚到这个保存点。

Begin； insert into test01 values(1); savepoint my\_savepoint01;

Inser into test01 values(2);insert into test01 values(2); rollback to savepoint my\_savepoint01;

Commit;

#### 6.11.6两阶段提交

Set max\_prepared\_transactions=10;

Create table testtab01(id int primary key);

Begin;

Insert into testtab01 values(1);

Prepare transaction ‘osdba\_global\_trans\_0001’;

Pg\_ctl stop –D $PGDATA; pg\_ctl start –D $PGDATA;

COMMIT PREPARED ‘osdba\_global\_trans\_0001’; select \* from testtab01;

死锁的发生必须具备的四个条件：

互斥条件、请求和保持条件、不剥夺条件、环路等待条件

Select locktype, relation::regclass as rel, virtualxid as vxid, virtualtransaction as vxid2, pid, mode, granted from pg\_locks where pid = 8464;

规则系统是通过查询重写来实现的，而触发器通常是为每一个行都触发执行一次，所以对于批量操作，如果使用规则，可能会生成更好的执行计划，从而提高效率。

## 15.Slony-I的使用

集群

## 16.Bucardo的使用

在postgresql中实现双向同步的软件，可以实现postgresql数据库的双master方案，

Bucardo的同步通过触发器来记录变化，并利用postgresql中的“notify”消息事件通知机制实现高效同步。

## 17.PL/Proxy的使用

能在postgresql数据库中实现数据水平拆分的软件。

18.pgpool-II的使用

连接池、复制、负载均衡、限制超过限度的连接、并行查询

18postgresql-xc的使用

基于postgresql数据库实现的真正的数据水平拆分的分布式数据库。