PostgreSQL Notes

Author : Yunqiu Xu

`Database` `SQL` `PostgreSQL`

Part I Tutorial

Chapter 1 Installation and configuration

1.1 Installation PostgreSQL 9.4.8 on Ubuntu 16.04

- Find `/etc/apt/sources.list.d/pgdg.list`, and add a line for the repository

```
deb http://apt.postgresql.org/pub/repos/apt/ YOUR_UBUNTU_VERSION_HERE-pgdg main
```

- Import the repository signing key, and update the package lists

```
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | \sudo apt-key add -
sudo apt-get update
```

- Then install it

```
apt-get install postgresql-9.4
```

- 安装完成后postgresql在系统中自动启动,可以运行以下命令查看状态

```
sudo /etc/init.d/postgresql status # 查看状态
sudo /etc/init.d/postgresql start #启动
sudo /etc/init.d/postgresql stop #停止
sudo /etc/init.d/postgresql restart #重启
```

1.2 Basic commands

- 登录默认用户postgres并修改密码:

```
sudo -u postgres psql #使用psql客户端登录,提示符变为"postgres=#"
ALTER USER postgres WITH PASSWORD 'asshole'; #修改密码,返回"ALTER ROLE"
\q  #退出
```

- 创建数据库: `sudo createdb dbname`

```
sudo createdb venturer
sudo -u venturer psql #登录该数据库
```

- 修改数据库：

```
#修改指定数据库名称:
alter database venturer rename to asshole;
```

- 删除数据库：`dropdb`

```
sudo dropdb test
```

- 登录UNSW数据库服务器

```
ssh zid@grieg.cse.unsw.edu.au

# create a new directory (/srvr/YOU/) to hold your server files
# create a unique and permanent IP address for your virtual host
priv srvr

#set the environment
source /srvr/zid/env

#opeartions about the database
pg start
...
pg stop
exit
```

- 修改系统登录密码:

```
sudo passwd -d venturer #密码过期信息已更改
sudo -u venturer passwd #重置密码
```

- 常用控制台命令

```
\h：查看SQL命令的解释，(e.g. \h select)
\?：查看psql命令列表
\l：列出所有数据库 (e.g. psql -l)
\c [database_name]：连接其他数据库
\d：列出当前数据库的所有表格(e.g. mydb \d)
\d [table_name]：列出某一张表格的结构
\du：列出所有用户
\e：打开文本编辑器
\conninfo：列出当前数据库和连接的信息
```

Chapter 2 The SQL Language

2.1 CREATE/DROP

```
CREATE TABLE weather (
    city            varchar(80),
    temp_lo         int,           -- low temperature
    temp_hi         int,           -- high temperature
    prcp            real,          -- precipitation
    date            date
);
DROP TABLE table_name;
```

2.2 INSERT INTO

```
#默认顺序
INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25, '1994-11-27'
);

#顺序可以颠倒
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
    VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');
```

- copy: 适合大批量数据,比INSERT INTO更优化

```
COPY weather FROM '/home/user/weather.txt';
```

- 从外部文件导入

```
#外部
psql venturer -f schema.sql

#内部
\i data.sql
```

2.3 SELECT

```
#检索全部
SELECT * FROM weather;

#检索部分columns
SELECT city, temp_lo, temp_hi, prcp, date FROM weather;

#计算并创建新列
SELECT city, (temp_hi+temp_lo)/2 AS temp_avg, date FROM weather;

#给出过滤条件
SELECT * FROM weather
    WHERE city = 'San Francisco' AND prcp > 0.0;

#排序
SELECT * FROM weather
    ORDER BY city,temp_lo;
```

- 以上都默认对某一个表进行操作，如果数据库中不止一个表需要`FROM`

```
SELECT DISTINCT city
    FROM weather
    ORDER BY city;
```

2.4 JOIN

- 内部联结：两个表共有的属性（交集）

```
#显示已经被分配部门的员工
SELECT *
FROM   employee
       INNER JOIN department
          ON employee.DepartmentID = department.DepartmentID

#等价于
SELECT *
FROM   employee, department
WHERE  employee.DepartmentID = department.DepartmentID
```

- 外部联结：可以匹配不同的属性
  - 左联结`LEFT OUTER JOIN`：匹配A所有符合的条目，如果B中没有则显示为NULL而非不输出
  - 右联结`RIGHT OUTER JOIN`：与左联结刚好相反
  - 全联结`FULL OUTER JOIN`:AB的并集而非交集(显示所有员工ID与部门ID,包括没任务的员工和没人的部门)

```
#显示所有雇员ID即使其未被分配部门
SELECT *
FROM   employee  LEFT OUTER JOIN department
          ON employee.DepartmentID = department.DepartmentID

#等价于
SELECT weather.city, weather.temp_lo, weather.temp_hi,
       weather.prcp, weather.date, cities.location
    FROM weather, cities
    WHERE cities.name = weather.city;
```

2.5 Aggregate functions

```
SELECT max(temp_lo) FROM weather;
```

- note that aggregate functions should not occur in
- Solution: 在WHERE子句中嵌套SELECT

```
SELECT city FROM weather
    WHERE temp_lo = (SELECT max(temp_lo) FROM weather);
```

- 多子句

```
SELECT city, max(temp_lo)
    FROM weather
    WHERE city LIKE 'S%'(1)
    GROUP BY city
    HAVING max(temp_lo) < 40; #分组后用HAVING过滤
```

2.6 UPDATE

```
UPDATE weather
    SET temp_hi = temp_hi - 2,  temp_lo = temp_lo - 2
    WHERE date > '1994-11-28'; #重新赋值
```

2.7 DELETE

```
DELETE FROM weather WHERE city = 'Hayward'; #删除某行
DELETE FROM table_name; #删除某表的所有行
```

- 注意区分和删除表的区别， 这里只是删除行返回空表， 表还存在

Chapter 3 Advanced Features

3.1 Views

- 有关视图的内容可以参考MySQL必知必会+SQL初级语法整理 CHAPTER 14
- VIEW是虚拟的TABLE，不包含数据，只包含使用时动态检索数据的查询

```
CREATE VIEW myview AS #这之后就是视图包含的查询子句
    SELECT city, temp_lo, temp_hi, prcp, date, location
        FROM weather, cities
        WHERE city = name;

SELECT * FROM myview;
```

3.2 主键/外键/索引

名称	定义	作用	个数
主键	唯一标识一条记录，不能有重复或为空	保证数据完整性	一个
外键	外键是另一表的主键, 可重复可空	与其他表建立联系	多个
索引	没有重复值，但可以有一个空值	优化查询排序	多个,但不可重复

```
CREATE TABLE cities (
        city     varchar(80) primary key,
        location point
);

CREATE TABLE weather (
        city      varchar(80) references cities(city),
        temp_lo   int,
        temp_hi   int,
        prcp      real,
        date      date
);
```

3.3 Transactions

- 可以参考MySQL必知必会+SQL初级语法整理CHAPTER 18

```
BEGIN;

UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice';
SAVEPOINT my_savepoint;

UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob';
-- oops ... forget that and use Wally\'s account
ROLLBACK TO my_savepoint;

UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Wally';
COMMIT;
```

3.4 Window functions

- 类似aggregate function,都是对指定数据做计算

- 区别: window function不会导致计算后的多行数据被分组为一行输出,而是保持原样

```
SELECT depname, empno, salary, avg(salary)
OVER (PARTITION BY depname)
FROM empsalary;
```

```
  depname  | empno | salary |          avg
-----------+-------+--------+-----------------------
 develop   |    11 |   5200 | 5020.0000000000000000
 develop   |     7 |   4200 | 5020.0000000000000000
 develop   |     9 |   4500 | 5020.0000000000000000
 develop   |     8 |   6000 | 5020.0000000000000000
 develop   |    10 |   5200 | 5020.0000000000000000
 personnel |     5 |   3500 | 3700.0000000000000000
 personnel |     2 |   3900 | 3700.0000000000000000
 sales     |     3 |   4800 | 4866.6666666666666667
 sales     |     1 |   5000 | 4866.6666666666666667
 sales     |     4 |   4800 | 4866.6666666666666667
(10 rows)
```

```
SELECT depname, empno, salary, rank()
OVER (PARTITION BY depname ORDER BY salary DESC)
FROM empsalary;
```

3.5 Inheritance

```
CREATE TABLE cities (
  name       text,
  population real,
  altitude   int     -- (in ft)
);

CREATE TABLE capitals (
  state      char(2)
) INHERITS (cities);
```

Part II

- 参考资料: PostgreSQL Documentation