# 一些几个经典的postgresql的sql语句

1.强制类型转换:

```
select created_at::date from user;
select created_at::varchar from user;
select cast(created_at as date) from user;
```

2.伴随语句查询

```
with tmp as (select name,age from user),
     tmp2 as  (select sal,join_at from company)
select * from user_detail,tmp,tmp2 where user_age = tmp.age
```

3.批量插入

```
insert into user(name,age) select name,age from account
```

4.内嵌条件语句

```
select case when t.result='success' then 0 else 1 end from table_result as t
```

5.无返回值且可以重复运行的函数

```
--将account中在t这一天时间里新增的用户同步到user表里
create function synchronize_user(t date) returns void as
$$
begin
delete from user where created=t;
insert into user(name,age) select name,age from account where created=t;
end;
$$
LANGUAGE 'plpgsql' VOLATILE;
```

调用方式:

```
select synchronize_user('2018-08-17')
```

6.函数名重载

```
--将account中所有的用户同步到user表里,可以与5并存
create function synchronize_user() returns void as
$$
begin
delete from user;
insert into user(name,age) select name,age from account;
end;
$$
LANGUAGE 'plpgsql' VOLATILE;
```

调用方法:

```
select synchronize_user()
```

7.创建视图

```
create view tmp as
 (
 select user_id,created,sum(total_money)
 from orders
 group by user_id,created
 )
```

使用时:

```sql
select * from tmp where created='2018-09-02'
```

8.自定义递增序列及使用

```sql
create sequence task_id_auto start with 12 increment by 1 no minvalue no maxvalue cache 1;
``
```sql
-- 创建时使用序列
create table t(
id integer default nextval('index_sequence')
)
-- 假设部分id被非法植入null,需要用序列进行重拍，显然不能删表重创
update t set id=nextval('index_sequence') where id is null
```

9.伴随语句与批量插入同时进行

```sql
with tmp as (
select user_id,sum(money) as money from users group by user_id
)
insert into money_analysis(user_id,money) select * from tmp
```

10.左右全连接出现的null处理补0或者补’’

```sql
--student的id比class_student的student的id多，造成了一些补充数据null
--对fee补0，对introduction补''
select student.id,
    case when fee is null then 0 else fee end ,
    case when introduction is null then '' else introduction
from student left join class_student on student.id = class_student.student_id
```

11. 返回table的函数

```sql
/*注意点
12. table声明的参数数量和类型要和内部的select一致,但取名不能一致，不然会冲突。
13. 调用时，对于插入修改删除类型的可以使用select fun(),但是对于select最好使用select * from fun()，因为select fun()对select查询会返回出raw message,
比如(1,'fw',9)而不是 user_id|name|age \n 1|'fw'|9
*/
create function find_master() returns table(nm varchar,uid integer,ag integer) as
$$
begin
return query
(
 select name,user_id,age from tuser
);
end;
$$
language 'plpgsql' volatile;
```

使用时:

```sql
select * from find_master()
-- 千万不要使用,这是数据流
select find_master()
```

14. 逻辑取反

并没有查到官方的取反操作,只有自己写了
```

```
create or replace function getnot(arg boolean) returns boolean as
$BODY$
begin
 if arg=true then
  return false;
 else
  return true;
 end if;
end;
$BODY$
language 'plpgsql' volatile;
```

使用时:

```
-- 选出用户表里，排除掉 id >5 且age < 3 的用户
select * from tuser where getnot(id>5 and age<3)
```

15.数组类型如何定义

```
// 对已有表增加字符串数组类型的字段
alter table message add column to_users_id varchar[] not null default array[]::varchar[]
// 新建表，字段为字符串数组类型
create table arr(
    message varchar[] not null default array[]::varchar[]
)
```

16. 如何对数组字段索引，并执行select操作，条件为’ 包含某元素’

```
-- 创建索引
create index gin_index_on_to_users_id on message using gin (to_users_id)
-- 准备数据
update message set to_users_id = array['1','2','3'] where id =5
-- 查找
select * from message where to_users_id::text[] @> array['3']
```

17. 查看数据库连接数和连接用户

```
-- 查看活跃用户
select * from pg_stat_activity;
-- 连接数
select count(*) from pg_stat_activity;
```

18. 修改数据表所属者:

```
alter table user_click owner to wechart
```

19. 把某个数据库的所有表的所有权限都给某个用户
    先以超级用户postgres 进入mydb数据库，再执行下面的语句，则xiaoming可以对mydb干一切事情。

```
\c mydb
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO xiaoming
```

20. 创建索引与删除索引

```
CREATE INDEX user_id_index ON usertable (user_id);
drop index user_id_index
```

21. 迁移一些表

只迁移insert数据,只有insert 语句

```
pg_dump -U zhangsan -d game --inserts  --column-inserts --data-only  -t game_order -t xyx_user_error> /tmp/game_order.sql;
psql -U zhangsan -d game_test < /tmp/game_order.sql
```

在两个数据库表结构相同的情况下只迁移全部数据

```
pg_dump -U zhangsan -d game --inserts  --column-inserts --data-only > /tmp/xyx_srv.sql;
psql -U zhangsan -d game_test < /tmp/game_order.sql;
```

迁移整个表

```
pg_dump -U zhangsan -d game  -t game_order> /tmp/game_order.sql;
psql -U zhangsan -d game_test < /tmp/game_order.sql
```

22.pgdump是整表操作无法指定条件，迁移where条件的数据，可以使用copy与\copy

```
-- 导出
\copy (select * from vx_template_user where user_id=32150123) to '/tmp/insert.db';
-- 导入
\copy vx_template_user from '/tmp/insert.db'
```

23.切换数据库

```
\c game
```

24.查看表的列类型和声明

```
\d user_table
```

25. 查看正在运行的sql与如何终止

```
SELECT
procpid,
start,
now() - start AS lap,
current_query
FROM
(SELECT
backendid,
pg_stat_get_backend_pid(S.backendid) AS procpid,
pg_stat_get_backend_activity_start(S.backendid) AS start,
pg_stat_get_backend_activity(S.backendid) AS current_query
FROM
(SELECT pg_stat_get_backend_idset() AS backendid) AS S
) AS S
WHERE
current_query <> '<IDLE>'
ORDER BY
lap DESC;
```

```
-- 参数是上条语句的procpid值
select  pg_terminate_backend(12928)
```

26. 创建json索引，添加json数据后，通过json查询

```
create table test_json(
    id serial primary key,
    data jsonb
);
-- 两种索引方式，后者快，大
create index idx_for_json on test_json using gin(data jsonb_path_ops);
-- create index idx_test2 on test2 using gin(doc);
insert into test_json(data) values('{"name": "ft"}');

select * from test_json where data::jsonb->>'name' = 'ft';
select data->> 'name' as name from test_json;
```

使用go语言动态查询json

```
engine = engine.Where(fmt.Sprintf("attach::json->>'%s' = '%v'", k, v))
```

26.1 对数组json进行查询

```
create table test_json(
    id serial primary key,
    data jsonb
);
insert into test_json(data) values('[{"name": "ft"},{"name": "uk"}]');

select * from (
    select id, json_array_elements(data) as data from test_json
) t
where data->>'name'='ft';
```

26.2 对json进行修改

```
update vx_subscribe_user set payload='{"time2": {"value": "2020年09月22日 12:00"}, "thing1": {"value": "您有福卡任务可完成,集福卡兑百元红包"}}'::jsonb;
```

运行时间分析

`explain analyze verbose select * from test`

跨表更新

```
update tmp set state = tmp2.state from tmp2 where tmp.id=tmp2.id
```

29. golang 使用in

```
db.Raw("select k from xxx where k in (?)", []interface{}{1, 2, 3}).Scan(&name)
```

30. 更新并返回更新前的数据，比如发放优惠码

```
with tmp as (
  select id,payload,used,payload_attach from one_off_coupon where used=1 and tag='ldj_1' limit 1
)
update one_off_coupon set used=2 ,put_on_at=now() from tmp where one_off_coupon.id = tmp.id returning tmp.*;
```

31. 一条语句事务执行多个东东

选出100条卡券，设置为已使用。

该条语句，支持多个服务，同时竞争

```
with tmp as (
  select id,payload,payload_attach from one_off_coupon where used=1 and tag='ldj_1' limit 100
), tmp2 as (
 update one_off_coupon set used=2 ,put_on_at=now() from tmp where one_off_coupon.id in (select id from tmp)
)
select * from tmp
```

32. 一个数据同步脚本

- 该脚本将game数据库的prop_config表，同步到game_test中

```
psql -U postgres -d game_test
123
delete from prop_config;
\q

pg_dump -U postgres -d game  -t prop_config> E:\tmp\prop_config.sql
123
psql -U postgres -d game_test < E:\tmp\prop_config.sql
123

psql -U postgres -d game
123
\c game_test
select count(1) from prop_config;

\q
```

33. 数据备份和恢复

```
-- 备份game数据库,以用户名postgres，随后需要输入密码
pg_dump -Fc -U postgres game > game.dmp

-- 跨服务器备份
pg_dump -Fc -h x.x.x.x -p 5432 -U zhangsan game > game_2020-05-12.dmp



-- 恢复
pg_restore -d game game.dmp

-- 跨服务器恢复
pg_restore -h x.x.x.x -p 5432 -U zhangsan -d game game.dmp
```

34. 使用触发器，更新某个表的updated_at

```
-- 该函数，整个数据库只需要执行/存在一个
CREATE OR REPLACE FUNCTION update_column_updated_at()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = now();
  RETURN NEW;
END;
$$ language 'plpgsql';

-- 假定需要对user_info支持updated_at自动更新
 CREATE TRIGGER tri_updated_at_refresh_on_user_info
   BEFORE UPDATE
   ON user_info FOR EACH ROW
   EXECUTE PROCEDURE update_column_updated_at();
```

35. 更新最佳序列自增值

```
with tmp as (
select max(id)+1 as max_id from prop_tag_config
)
select setval('prop_tag_config_id_seq', tmp.max_id) from tmp;
```