视图就是由查询语句定义的虚拟表。 对用户来说， 视图就如同一张真的表。 从视图中看到的数据可能来自数据库中的一张或多张表， 也可能来自数据库外部， 这主要取决于视图的查询语句是如何定义的。

**视图的优点：**

1.可使复杂的查询易于理解和使用。

2.安全原因， 视图可以隐藏一些数据， 如在一张用户表中， 可以通过定义视图把密码等敏感字段去掉。

3.把一些函数返回的结果映射成视图。

一般数据库提供的视图大多是只读的， PostgreSQL数据库中创建的视图是可更新的。

**视图管理**

**创建视图**

语法格式：

CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column\_name [, ...] ) ]

AS query

示例：

1.创建一张users表， 其中包含了敏感字段 password， 现在要创建一个视图把password排除

create table users(id int,name varchar(40),password varchar(256),age int);

create view v\_user as select id, name, age from t\_user;

select \* from v\_user;

2.使用TEMP或TEMPORARY关键字创建一个临时视图， 当session结束时， 这个视图就会消失：

create temp view v\_user as select id, name, age from users;

insert into users values(1,'zhangsan','123456', 22);

select \* from v\_user;

完成上述操作后退出， 再打开后使用“\d”命令就看不到该临时视图了

3.为查询列定义一个名称。

create view v\_user(user\_id,user\_name,user\_age) as select id, name, age from users;

**修改视图：**

修改视图必须要用replace，意思就是替换

CREATE OR REPLACE VIEW view\_name AS query

create or replace view v\_user as select name,age from users;

**重命名视图：**

alter view v\_user rename to v\_user2;

**删除视图：**

drop view [if exists] 视图名;

if exists就是一句判断，是否存在，可以不写

**可更新视图**

PostgreSQL9.3以上的版本中创建的简单视图默认是可以更新的，也就是说可以使用 INSERT 、UPDATE 和 DELETE命令在该视图中更改数据了。

满足以下条件的视图是可更新的：

定义查询不得包含以下子句之一：GROUP BY，HAVING，LIMIT，OFFSET，DISTINCT，WITH，UNION，INTERSECT和EXCEPT。

选择列表不得包含任何窗口函数，任何返回集合的函数或任何聚合函数，例如SUM，COUNT，AVG，MIN和MAX。

可更新视图可以同时包含可更新列和不可更新列。如果尝试插入或更新不可更新的列，PostgreSQL将引发错误。

当执行诸如INSERT，UPDATE或DELETE之类的操作时，PosgreSQL会将此语句转换为基础表的相应语句。

如果在视图的定义查询中有WHERE条件，仍然可以更新或删除在视图中不可见的行。但是，如果要避免这种情况，可以在定义视图时使用WITH CHECK OPTION。

执行更新操作时，必须在视图上具有相应的特权，但不必在基础表上具有特权。但是，视图所有者必须具有基础表的相关特权。

**示例：**

1.创建一个可更新视图

create view v\_user as select id, name, age from t\_user where id <= 10;

2.使用视图插入记录：

insert into v\_user values(21,'zhangsan',22);

3.通过视图删除记录：

delete from v\_user where id=1;

4.通过视图更新记录，但不会更新任何记录，因为视图仅包含 id 在1到10的数据：

update v\_user set age = 22 where id = 12;

5.通过视图更新记录，修改后的数据不再落在视图可见范围内，也就是说视图更新后少了一条记录。

update v\_user set id = 13 where id = 1;

**WITH CHECK OPTION子句的使用**

当使用with check option 子句创建视图的时候，PostgreSQL会通过视图检查正在更改的每一行，以使其符合视图的定义。

为了防止用户插入或更新在视图中不可见的行，创建视图时需要使用WITH CHECK OPTION子句。

create view v\_user as select id, name, age from t\_user where id <= 10 with check option;

update v\_user set id = 14 where id = 2;--ERROR

insert into v\_user values(21,'lisi',20);--ERROR

注意：with check option 对于delete操作没有影响

delete from v\_user where id =   21;--SUCCESS

对于没有where字句的视图，使用with check option是多余的

create view v\_user as select id, name, age from users with check option;

update v\_user set id = 14 where id = 2;--SUCCESS

delete from v\_user where id =   21;--SUCCESS

with check option选项总结：

1.update，要保证数据update之后能被视图查询出来，也就是要符合where的条件

2.insert,保证insert的数据能被视图查询出来

3.delete,有无 with check option都一样

4.对于没有where字句的视图，使用with check option是多余的

**用LOCAL和CASCADED检查的范围**

使用LOCAL，将检查视图WHERE子句，但不检查基础视图。

使用CASCADED，检查视图WHERE子句，然后检查递归到基础视图，并向它们添加WITH CASCADED CHECK OPTION（为了检查;它们的定义保持不变），并应用相同的规则。

创建视图

create view v\_user1 as select id, name, age from t\_user where id <= 10;

创建另一个视图，该视图基于v\_user1视图

create view v\_user2 as select id, name, age from v\_user1 where id >= 5 with cascaded check option;

通过v\_user2视图插入一条记录

insert into v\_user2 values(1,'zhangsan',22);--ERROR

即使 v\_user1视图没有WITH CHECK OPTION子句，也违反了v\_user1视图的定义条件。使用WITH CASCADED CHECK OPTION时，PostgreSQL会检查v\_user2视图以及所有基础视图的视图定义条件。

使用WITH LOCAL CHECK OPTION：

create or replace VIEW v\_user2 as select id, name, age from v\_user1 where id >= 5 with local check option;

insert into v\_user2 values(11,'zhangsan',22);-SUCCESS

插入成功，因为新行满足了v\_user2视图的定义条件。PostgreSQL没有检查基本视图的定义条件。

**物化视图**

物化视图(Materialized View)和视图(View)类似，不同之处在于：物化视图把结果集保存在起来，查询的时候直接读取保存的结果集，而不必扫描原始表。物化视图的结果集的存储和扫描方式和普通表的方式相同。

**语法格式：**

CREATE MATERIALIZED VIEW view\_name

AS query

WITH [NO] DATA;

如果要在创建时生成数据，使用WITH DATA选项，否则使用WITH NO DATA，该视图被标记为不可读。这意味着您法从视图中查询数据，除非将数据加载到视图中。

**示例1：**

1.创建物化视图

create materialized view mv\_user as select age,count(\*) from users group by age;

2.更新数据：

PostgreSQL目前不支持物化视图的自动/增量更新。

refresh materialized view mv\_user;

3.创建索引：

物化视图是支持索引的。

create unique index on mv\_user (age);

4.物化视图不支持数据插入

insert into mv\_user values (...); --ERROR

**示例2：**

1.使用了WITH NO DATA选项

create materialized view mv\_user as select age,count(\*) from users group by age with no data;

select \* from mv\_score;--ERROR

2.更新数据

refresh materialized view mv\_user;

再次查询数据

select \* from mv\_user;-SUCCESS

3.并发更新数据

更新物化视图，PosgreSQL会锁表，可以使用CONCURRENTLY选项来避免这种情况：

首先为视图创建一个UNIQUE索引。

create unique on mv\_user(age);

然后更新物化视图

refresh materialized view concurrently mv\_user;

**示例3**

物化视图和外部表配合使用

物化视图允许通过外部数据包装器对来自外部的数据进行更快的访问。由于本地系统上可以使用高速缓存，因此比起访问一个远程系统的性能更好。鉴于file\_fdw不支持索引，可以在物化视图上放置索引。

copy (select \* from t1 limit 10) to '/tmp/t1.csv'; --生成外部文件csv

create extension file\_fdw;

create server local\_file foreign data wrapper file\_fdw;

create foreign table fdw\_t1 words (id int,name text,age int)

  server local\_file

  options(filename '/tmp/t1.csv');

create materialized view mv\_t1 as select \* from fdw\_t1;

create index on mv\_t1(id);

select \* from mv\_t1 where id =10;

**示例4**

物化视图和远程表配合使用

假设PostgreSQL中定义了一个外表(Foreign Table)，该表存在于外部的一个数据库。扫描该表时，可能会很慢。

如果针对这个外表定义一个物化视图，就可以把对外表的查询结果保存起来，后续的扫描就不必扫描并加载外部数据了。

1.创建远程表插件

create extension postgres\_fdw;

2.创建server remote\_table封装192.168.1.102主机，5432端口，testdb数据库。

create server remote\_table foreign data wrapper postgres\_fdw options(host '192.168.1.102',port '5432',dbname 'testdb');

3.建立user mapping

CREATE USER MAPPING指定连接远程数据库的用户。

create user mapping for postgres server remote\_table options(user 'postgres',password 'postgres');

4.创建外部表

注意：声明的列数据类型必须要匹配实际的远程表，列名也必须匹配。

create foreign table rmt\_t1(id int,name text,age int) server remote\_table options(schema\_name 'public', table\_name 't1');

5.创建物化视图

create materialized view mv\_t1 as select \* from rmt\_t1;

6.创建索引

create index on mv\_t1(id);

**总结**

1.使用PostgreSQL时，如果一个视图的查询耗时很大，可以考虑使用物化视图。因为物化视图把结果集以表的方式记录下来，下次查询时候，直接使用顺序扫描返回结果。

2.如果原始表的数据更新频繁，而物化视图刷新很慢，则不太适合使用物化视图。

3.最终是否要使用物化视图，以及如何使用物化视图，要结合具体情况决定。

**物化视图管理**

重命名物化视图

alter materialized view mv\_t1 rename to mv\_t2;

删除物化视图

drop materialized view view\_name;