**视图是通过自定义查询 SQL 生成的虚拟表、逻辑表，作为 select 语句保存在数据字典中，本身并不包含数据。**

下面通过一个简单示例来了解视图：

*---- 下面是一段有子查询的 SQL，---- 假设这段子查询需在除本次查询以外的各种场景使用，那么我们可以为其创建视图***select** a.last\_name, a.product\_id**from**(

**select** people.id, people.last\_name, cart.product\_id

**from** people

**join** cart **on** cart.pepople\_id **=** people.id

**where** first\_name **=** 'join' ) a **where** a.id **=** 3;

*---- 创建视图***create** **view** shorter\_join **as**

**select** **\***

**from** people

**join** cart **on** cart.pepople\_id **=** people.id

**where** firstname **=** 'join';

*---- 视图创建后，可直接使用***select** last\_name, product\_id**from** shorter\_join **where** id **=** 3;

你看，视图用起来就是那么简单。那视图一般的用途、优点都有哪些呢？

**1）使用视图，简化数据查询操作，提高效率：**使用视图的用户完全不需要关心视图内部对应的逻辑（各种关联、筛选条件），这样一个频繁使用的、复杂的 SQL 就不需要反复写多次。比如上述示例；

**2）使用视图，基表的数据就有了安全性：**使用视图的用户只能访问他们被允许查询的结果集。

**3）使用视图，可以保证数据独立：**一旦视图的结构确定了，可以屏蔽表结构变化对用户的影响，源表增加列对视图没有影响；源表修改列名，则可以通过修改视图来解决，不会造成对访问者的影响。

## **二、分析师可以用它做什么？**

上一节提到视图可以简化数据操作，分析师可以把常用的一些查询 SQL 逻辑创建成视图。

比如，优惠券的核销与订单的一一对应关系，可能你很多场景都需要把券和订单关联上，可以创建视图；

比如，近期某个分析专题需频繁查询某几个页面的访问、曝光、点击等行为埋点，你可以创建视图；

再比如，你所负责区域、城市的用户、商户列表构建成视图，如此你就不需要每次查询都写上区域、城市等等条件了。

当然，我想有同学会说我直接创建生成数据表也可以达到这样的效果。的确如此，构建数据中间表也是现在广大分析师的常用手段。什么情况用视图，什么情况用数据中间表？我建议，

**在轻量、数据定义仍需要经常变更的情况下，建议使用视图。**

原因在于，视图优势是易于实现， 需变更时代价小（只需修改视图创建 SQL 即可，中间表则需要修改表定义、变更写入代码等），三天两头的改动建议还是用视图。

**在数据定义稳定、且查询复杂耗时大的情况下，建议使用中间表。**

原因在于，视图并未生成物理存储查询压力仍在基表（生成视图的数据库表）；当数量大查询复杂时，使用中间表才可以提高查询速度，缩短查询时间。视图查询语句查一次要1小时还是建中间表。

明白了什么是视图，视图和中间表的差异，接下来，我们对视图的语法使用做一些简单介绍：

## **三、视图的使用语法**

**1）创建视图**

**CREATE** [**OR** **REPLACE**]

[ALGORITHM **=** {UNDEFINED **|** MERGE **|** TEMPTABLE}]

[**DEFINER** **=** { **user** **|** **CURRENT\_USER** }]

[**SQL** **SECURITY** { **DEFINER** **|** **INVOKER** }]**VIEW** view\_name [(column\_list)] **AS** select\_statement

[**WITH** [**CASCADED** **|** **LOCAL**] **CHECK** **OPTION**]

语法说明如下：

1. OR REPLACE：表示替换已有视图。
2. ALGORITHM：表示视图选择算法。
   1. UNDEFINED(未定义的)，由 MySQL 自动选择要使用的算法
   2. MERGE 将视图的语句与视图定义合并起来，使得视图定义的某一部分取代语句的对应部分
   3. TEMPLATE 将视图的结果存入临时表，然后使用临时表执行语句
3. DEFINER：指出谁是视图的创建者或定义者：definer= '用户名'@'登录主机'；如果不指定，则创建视图的用户就是定义者，等价于指定关键字CURRENT\_USER(当前用户)。
4. SQL SECURITY
   1. SQL SECURITY DEFINER：定义(创建)视图的用户必须对视图所访问的表具有select权限
   2. SQL SECURITY INVOKER：访问视图的用户必须对视图所访问的表具有select权限
5. SELECT\_STATEMENT：即 SELECT 语句。SELECT语句不能引用任何变量，包括局部变量，用户变量和会话变量。
6. 列名：视图名后面的列的数量必须匹配 SELECT 子句中的列的数量。
7. [WITH [CASCADED | LOCAL] CHECK OPTION]：表示视图在更新时保证在视图的权限范围之内。此处重点说明几个点：
   1. CASCADED 是默认值，表示更新视图的时候，必须满足所有针对该视图的所有视图条件才可以更新；LOCAL 表示更新视图的时候，只要满足本视图的条件就可以更新。

对于嵌套视图的场景，做DML操作时，cascaded在检查约束时将检查所有视图，local只检查将要更新的视图本身。推荐使用 WITH CASCADED CHECK OPTION选项，可以保证数据的安全性。  
  
对于申明了 with check option 的视图，对它所做的DML操作的结果，不能违反视图的WHERE条件的限制。



**2）视图查询：**像使用表一般使用即可。使用 show create view 语句可查看视图信息。

**3）视图更改：**

1. CREATE OR REPLACE VIEW语句修改视图定义
2. DML操作更新视图，因为视图本身没有数据，因此对视图进行的 DML 操作最终都体现在基表中。当然，视图的DML操作，不是所有的视图都可以做DML操作，有下列内容之一，视图不能做DML操作：
   1. select子句中包含distinct
   2. select子句中包含组函数
   3. select语句中包含group by子句
   4. select语句中包含order by子句
   5. select语句中包含union 、union all等集合运算符
   6. where子句中包含相关子查询
   7. from子句中包含多个表
   8. 如果视图中有计算列，则不能更新
   9. 如果基表中有某个具有非空约束的列未出现在视图定义中，则不能做insert操作
3. Drop 视图：drop view view\_name