触发器: 创建视图或者触发器，用完需要删除。

> - 触发器是一种特殊类型的存储过程，不由用户直接调用。创建触发器时会对其进行定义，以便在对特定表或列作特定类型的数据修改时执行。触发器可以查询其他的表，而且可以包含复杂的SQL语句他们主要用于强制服从复杂业务的规则或要求。
> - 触发器是与表相关的数据库对象，在满足定义条件时触发，并执行触发器中定义的语句集合。触发器的这种特性可以协助应用在数据库端确保数据库的完整性。

> - 触发器在数据库中定义了一系列的操作，可以在对指定表进行插入，更新或者删除的同时自动执行这些操作。
>   - 例如：西安北站所属办公室----->北站发一篇公告，数据库向西安北站添加一条记录---> 同时向办公室数据库中添加同样的记录。 这是执行了两条SQL语句
>   - 如果定义一个触发器（向西安北站添加一条记录的时候，同时添加同样的记录到办公室） 这样执行一条SQL语句。

触发器的优点：

> - 在数据库中的，不必编写每个触发器在应用程序（java）中执行的操作。
> - 开发更快，因为触发器是编写在数据库中的，不必编写每个触发器在应用程序（java）中执行的操作。
> - 更容易维护（不需要频繁的修改代码，减少成本），定义触发器后，访问目标表数据库会自动调用触发器。
> - 业务全局实现，如果需要修改业务，只需要修改触发器，不需要修改应用程序代码。

触发器的缺点

> 尽管触发器有很多优点，但是在实际的项目开发中，特别是OOP思想的深入，触发器的弊端也逐渐突显，主要：
>
> - 过多的触发器使得数据逻辑变得复杂
> - 数据操作比较隐含，不易进行调整修改
> - 触发器的功能逐渐在代码逻辑或事务中替代实现，更符合OO思想。
> - 建议谨慎使用触发器。

触发器的分类：

> SQL Server 包括两种常规类型的触发器：数据操作语言 (DML) 触发器和数据定义语言 (DDL) 触发器。 当INSERT、UPDATE 或 DELETE 语句修改指定表或视图中的数据时，可以使用 DML 触发器。 DDL 触发器激发存储过程以响应各种 DDL 语句，这些语句主要以CREATE、ALTER 和 DROP 开头。 DDL 触发器可用于管理任务.

> - 通常说的触发器就是DML触发器：DML 触发器在 INSERT、UPDATE 和 DELETE 语句上操作，并且有助于在表或视图中修改数据时强制业务规则，扩展数据完整性。

> - 前置触发器 ： 在对目标表进行 更新，插入之前执行。即在操作之前触发
> - 后置触发器 ： 对目标包进行 更新，插入，删除 操作之后执行
> - before delete 触发器: 在对目标表删除之前执行
> - insted of 触发器： 对复杂的视图 执行插入，更新和删除时执行。

触发器的作用

> - 触发器可通过数据库中的相关表实现级联更改；通过级联引用完整性约束可以更有效地执行这些更改。
> - 触发器可以强制比用 CHECK 约束定义的约束更为复杂的约束。与 CHECK 约束不同，触发器可以引用其它表中的列。例如，触发器可以使用另一个表中的 SELECT 比较插入或更新的数据，以及执行其它操作，如修改数据或显示用户定义错误信息。
> - 触发器还可以强制执行业务规则
> - 触发器也可以评估数据修改前后的表状态，并根据其差异采取对策。

触发器的使用:

```
   语法
   ----------------------------------------------------
       CREATE TRIGGER trigger_name
       ON {table_name | view_name}
       {FOR | After | Instead of} [insert, update,delete]
       AS
       sql_statement
   ----------------------------------------------------
   关键词 trigger
   ----------------------------------------------------
     create trigger  触发器名称  //创建触发器
     after   insert   on   table1    //触发器类型，什么时候启动触发器
     for  each  row  //受影响的行
     begin  //开始
     insert  into   table2(table2_id)  values(new.table1_id)；//  触发器要执行的
业务
     end;
```

- 删除触发器:

```
  drop  trigger  触发器名字;
```

视图

> 创建一张虚拟的表：少执行SQL语句，因为在和java连接的时候，每执行一条语句就会连接JDBC接口，消耗资源。

普通员工无法查看薪资，管理成可以查看，
SQL语句举例：

```
if( type = "employee" ){
       String sql = "select  id, name  from  employee";
      }

      if (tyepe = "manager"){
      String sql  =  " select  *  from employee";
      }
    String  sql = " select  *  from  employee";
```

最笨的办法是创建两张表，区别是一张有薪资　一张没有薪资字段。维护数据的时候，必须同时维护两张表。

- **什么是视图**：一个人包含某个查询的虚拟表

> - 对视图进行操作，依托于真实的表
>   - 主要目的简化语句
>   - 对性能没有改善
> - 视图允许嵌套
> - 视图不能索引，没有关联，没有默认值

- **视图的用途**

> - 筛选表中的行，降低数据库的复杂程度，
> - 防止未经许可的用户访问敏感数据，提高安全性
> - 将多个物理数据抽象为一个逻辑数据库
> - 一次编写多次使用
> - 可授权访问表的特定部分
> - 封装计算字段

- 视图的基本操作和语法
  - 创建视图

```
 CREATE VIEW view_name AS SELECT column_naem(s) FROM table_name WHERE condition
 ----------------------------------------------
 创建视图 ，关键字 view：create view 视图名 as select 字段 from 表名;
```

  - 更新视图

```
 CREATE OR REPLACE VIEW view_name AS SELECT table_name WHERE condition==
```

  - 撤销视图

```
 DROP VIEW view_name
```

- 使用视图

```
SELECT * FROM view_name
```

- 他是一条SQL查询语句
- 本身不包含数据
- 是一张虚表