国际物流云商系统第九天

一. 回顾

1.细粒度权限的控制

2.POI 报表

八个步骤

3.Excel 版本之间的区别

4.模板打印的步骤

1. 制作模板
2. 加载模板文件，得到工作簿
3. 获取工作表
4. 获取行对象
5. 获取单元格对象
6. 读取单元格的内容
7. 读取单元格的样式
8. 保存，关闭流
9. 下载

二．海量数据 POI 导出

1.海量数据的概念

海量：早期百万，大数据出来后，上亿
Hibernate 数据量单表要小于 500 万；
Mybatis/jdbc 单表数据大于 500 万，oracle 不要超过 1 亿

2.POI 实现百万数据 POI 分析

Excel2003 数据量有限制：65536 行 ,256 列
HSSFWorkbook 对象只能操作 excel2003 xls 扩展名
XSSFWorkbook 可以支持 excel2007 及以上的版本 xlsx 扩展名
Excel2007 支持的单 sheet： 1048576 行，16384 列
理论上 XSSFWorkbook 是可以实现百万级别数据报表。实质运行时，可能会产生问题！
原因是在执行 POI 报表时所产生的行对象，单元格对象，样式对象，字体对象，它们都不会销毁，只有 POI 报表完成时这些对象才会销毁，这就导致了堆内存中对象的个数急剧增加，最后可能导到内存溢出
SXSSFWorkbook 优点：
SXSSFWorkbook 可以支持百万级别数据的 POI

```
//Workbook wb = new HSSFWorkbook();//只能作用于excel2003
//Workbook wb = new XSSFWorkbook();//作用于excel2007及以上版本，虽然可以支持大量数据，但实际测试时会失败
Workbook wb = new SXSSFWorkbook();//可以解决百万数据POI，但不支持模板，要求POI的相关jar在3.0以上
```

它不支持打开模板文件
SXSSFWorkbook 缺点：

- SXSSFWorkbook(int rowAccessWindowSize) 参数代表内存中对象达到的个数

高版本的 poi 才有，3.0 之后的版本

3.POI 实现百万数据实现及原理分析

在初始化 SXSSFWorkbook 这个对象时，可以指定在内存中所产生后的 POI 导出相关对象的个数（默认为 100 个），一旦内存中对象的个数达到这个指定值时，就将内存中的这些对象的内容写入磁盘中（xml 文件格式），就可以将这些对象从内存中销毁掉，以后只要达到这个值时，都会这样处理。在最终 excel 导出完成时，也会将写入在 xml 文件中的内容一起导出。

缺点：内存与磁盘交互时，需要占用 IO 流相关操作，它本身也是耗时的，如果机器配置比较低时，就会出现磁盘写操作还没有完成，但是内存中又存在了指定数量的对象。
量变引起质变。

CSV 文本格式，带格式的 txt，excel 直接支持打开
需求：

将数据库百万数据导出到 excel 中。系统数据的备份（样式）

1.先将数据导出到 xls 格式的 excel 文件中
2.Invalid row number (65536) outside allowable range (0..65535)
百万数据可以分批次进行导出（分不同 sheet）,就可以很好避免堆内存溢出的问题

4.POI 百万数据的问题解决

使用测试类进行测试

```
Connection conn = DriverManager.getConnection(url, user,password);
Statement stmt = conn.createStatement();
String sql = "SELECT * FROM CONTRACT_PRODUCT_C";     //100万测试数据
ResultSet rs = stmt.executeQuery(sql);                                    //bug 要分次读取，否则记录过多
long  startTime = System.currentTimeMillis();    //开始时间
System.out.println("strat execute time: " + startTime);
int rowNo = 0;
int colNo = 0;
while(rs.next()) {

    for(int i=0;i<5000;i++){
        colNo = 0;
        nRow = sheet.createRow(rowNo++);

        nCell = nRow.createCell(colNo++);
        nCell.setCellValue(rs.getString(colNo));

        nCell = nRow.createCell(colNo++);
        nCell.setCellValue(rs.getString(colNo));

        if(rowNo%100==0){
            System.out.println("row no: " + rowNo);
        }
    }
    Thread.sleep(1);              //休息一下，防止对CPU占用
}
long finishedTime = System.currentTimeMillis(); //处理完成时间
System.out.println("finished execute  time: " + (finishedTime - startTime)/1000 + "s");
```

测试结果：

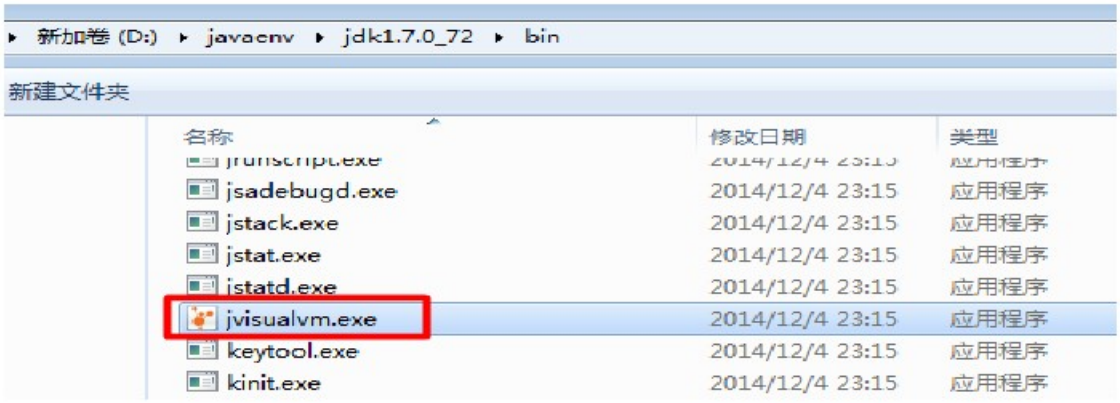
```
strat execute time: 1498461549027
finished execute  time: 1s
write xlsx file time: 4s
```

通过测试证明，我们可以 SXSSFWorkbook 来实现百万数据的导出，实际所消耗的时间是很少的。只要将最好的技术很好的结合在一起，比如 jdbc，oracle，SXSSFWorkbook，Thread 结合一起就可以很好的实现百万数据的 POI。

三．POI 百万数据的性能监视工具

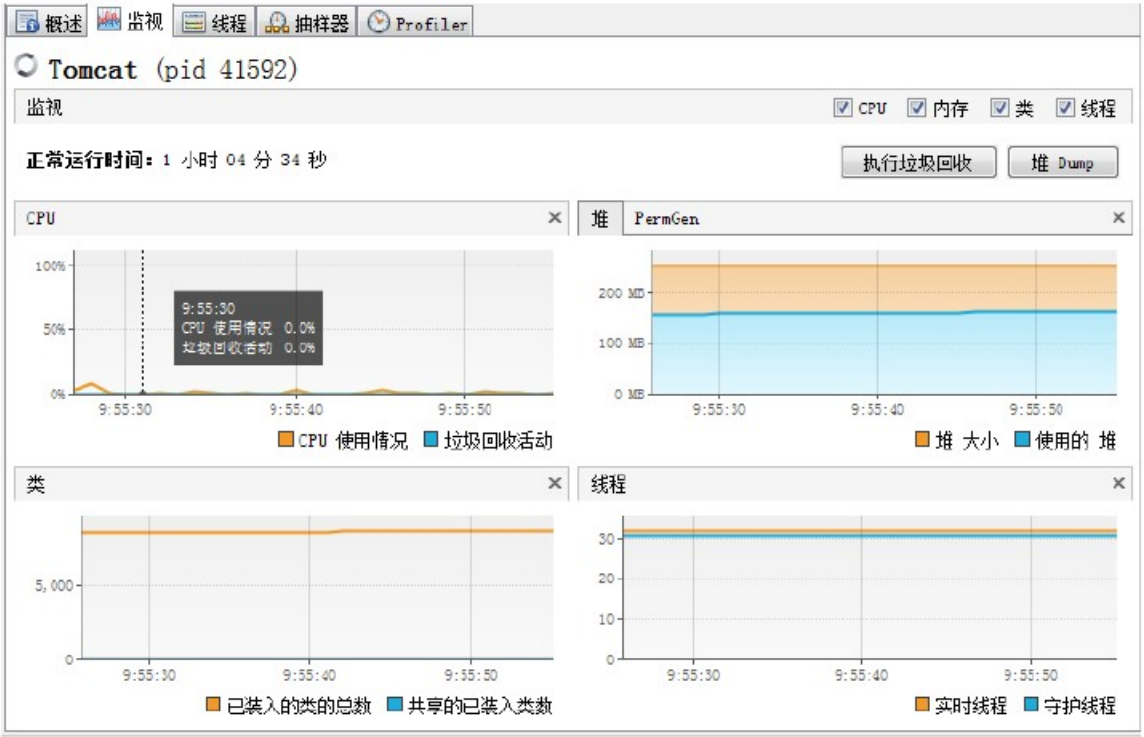
没有性能监视的一切推断都只是停留在理论上的分析过程，我们也可以使用 JDK 自带的性能监视工具来监视 CPU，垃圾回收器，堆内存的分配和使用情况，让我们的推断变得更加可视，从而证明我们的理论是可靠的。

1.JDK 自带的 jvisualvm 工具的位置

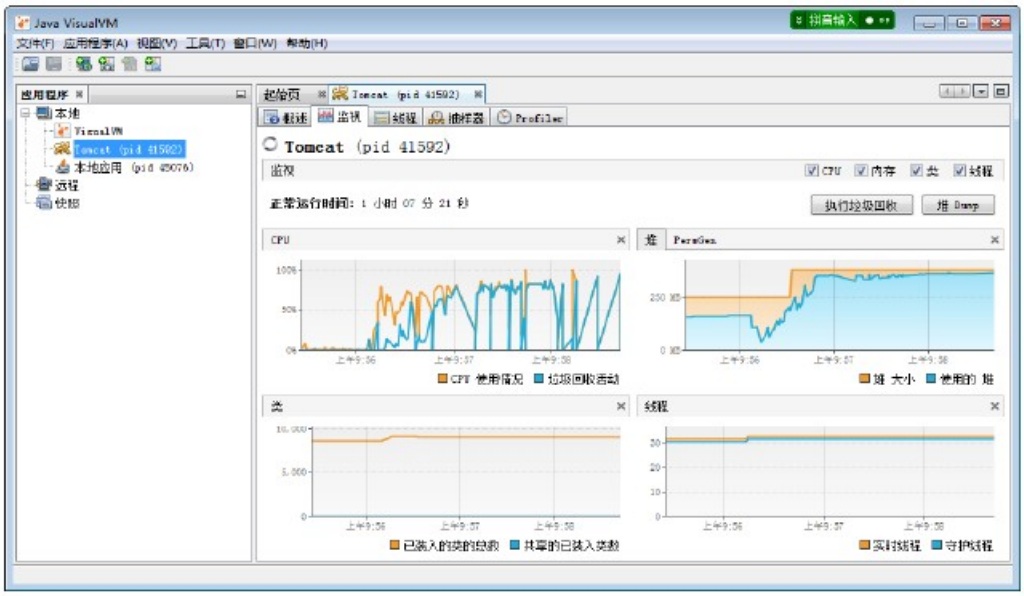


2.性能监视的过程

在刚打开的时候，我们可以看到 cpu,垃圾回收器，堆内存很平稳运行。



当运行百万数据 POI 的时候，开启 jvisualvm 监视工具，可以看到结果如下图。



为了解决问题，我们可以加大 **jdk** 配置的内存，**Tomcat** 的内存配置，但加大也是有限度的，这就跟我们的机器硬件配置有关了，所以这只是解决问题的一个办法，更好的做好是通过软件的手段来实现。

三．细粒度权限控制

1.管理本部门及下属部门权限分析

1.当事人查看自己的记录
2.部门经理查看当前部门下员工所添加的记录
3.总经理查看所有记录
4.查询当前部门及下属部门

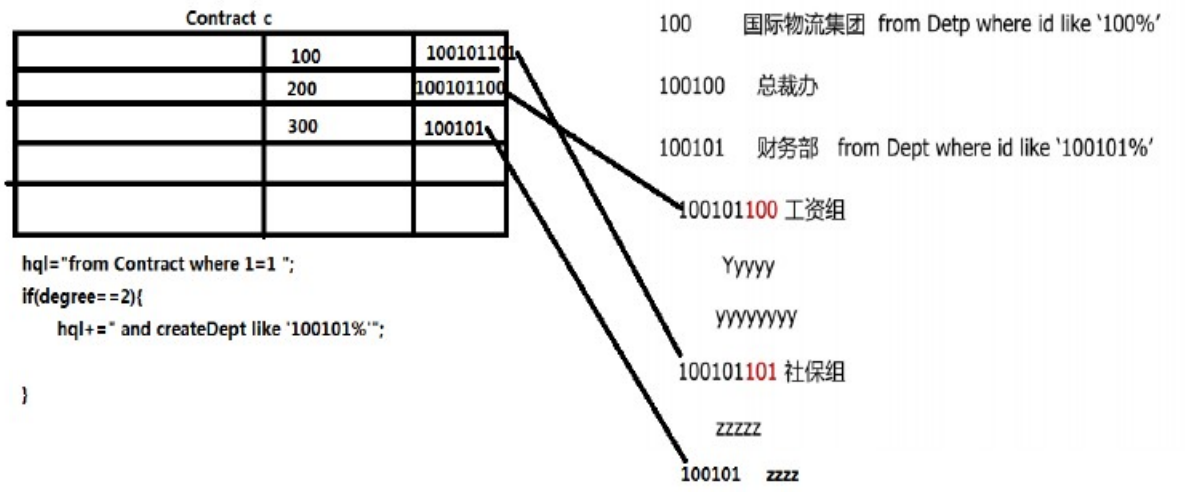
100 国际物流集团 **from Detp where id like '100%'**
100100 总裁办
100101 财务部 **from Dept where id like '100101%'**
100101100 工资组
Yyyyy
yyyyyyyy
100101101 社保组
zzzzz
from Contract where createDept in('100101', '100101100', '100101101')

from Contract where createDept like '100101%'

100102 船运部 **from Dept where id like '100102%'**
100102100 装箱部
100102101 委托部

100199　　救火队

2.实现过程



3.测试代码

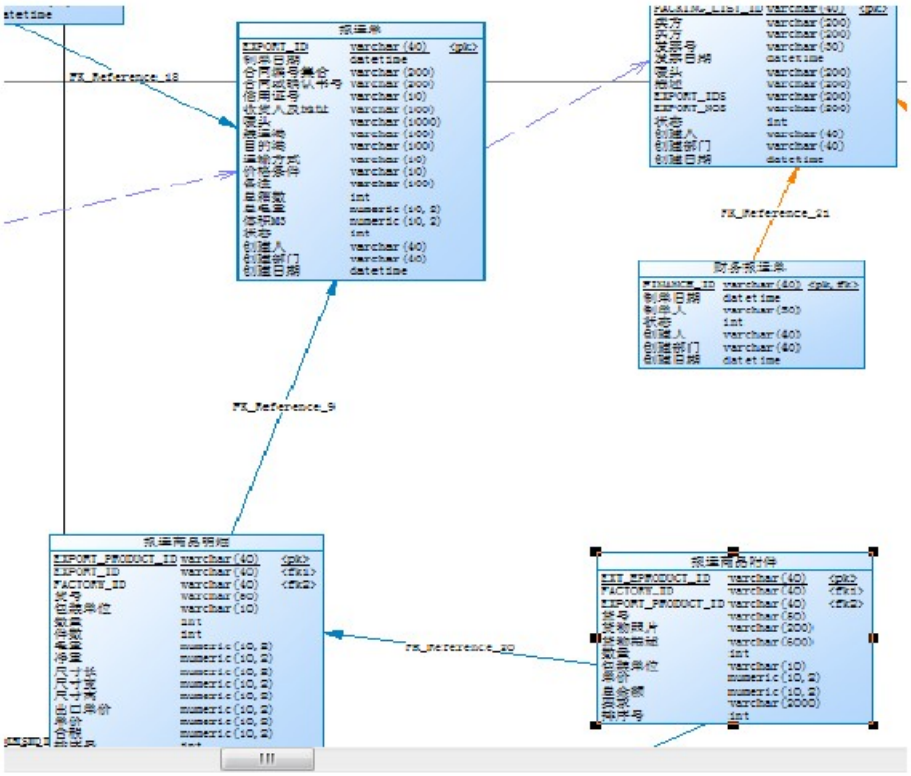
```
}else if(degree==2){
    //管理本部门及下属部门
    p = cb.like(root.get("createDept").as(String.class),user.getDept().getId()+"%");
```

四．出口报运模块

1.出口报运模块介绍

国际物流运输的货物到国外，这个过程需要经过海关人员的审批。所以公司对于出口的产品就要进行出口报运，就需要形成出口报运单，将来公司的报运人员，需要拿着出口报运单到海关进行审批。而出口报运单需要打印出来，打印的数据来自于数据库表。

2.出口报运模块表结构



一个报运单可以针对多个购销合同一起进行申报，并且只能针对购销合同 state 字段的取值为 1 的购销合同进行报

3.出口报运模块的 PO 类

Export 类

```
@Entity
@Table(name="EXPORT_C")
@DynamicInsert(true)
@DynamicUpdate(true)
public class Export implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name="EXPORT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;


    @OneToMany(mappedBy="export",cascade=CascadeType.ALL)
    @OrderBy("ORDER_NO")
    private Set<ExportProduct> exportProducts;   //报运下的货物 一对多

    @Column(name="INPUT_DATE")
    @Temporal(TemporalType.TIMESTAMP)
    private Date inputDate;                 //制单日期
```

ExportProduct 类

```
@Entity
@Table(name="EXPORT_PRODUCT_C")
@DynamicUpdate(true)
@DynamicInsert(true)
public class ExportProduct implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name="EXPORT_PRODUCT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @ManyToOne
    @JoinColumn(name="EXPORT_ID")
    private Export export;          //报运货物和报运的关系，多对一

    @ManyToOne
    @JoinColumn(name="FACTORY_ID")
    private Factory factory;        //报运货物和厂家的关系，多对一

    @OneToMany(mappedBy="exportProduct",cascade=CascadeType.ALL)
    private Set<ExtEproduct> extEproducts;      //报运货物和报运附件的关系，一对多
```

ExtEproduct 类

```
@Entity
@Table(name="EXT_EPRODUCT_C")
@DynamicUpdate(true)
@DynamicInsert(true)
public class ExtEproduct implements Serializable{
    @Id
    @Column(name="EXT_EPRODUCT_ID")
    @GeneratedValue(generator="system-uuid")
    @GenericGenerator(name="system-uuid",strategy="uuid")
    private String id;

    @ManyToOne
    @JoinColumn(name="EXPORT_PRODUCT_ID")
    private ExportProduct exportProduct;        //附件和货物，多对一

    @ManyToOne
    @JoinColumn(name="FACTORY_ID")
    private Factory factory;                    //附件和厂家，多对一

    @Column(name="PRODUCT_NO")
    private String productNo;
```

五. 打断设计思想

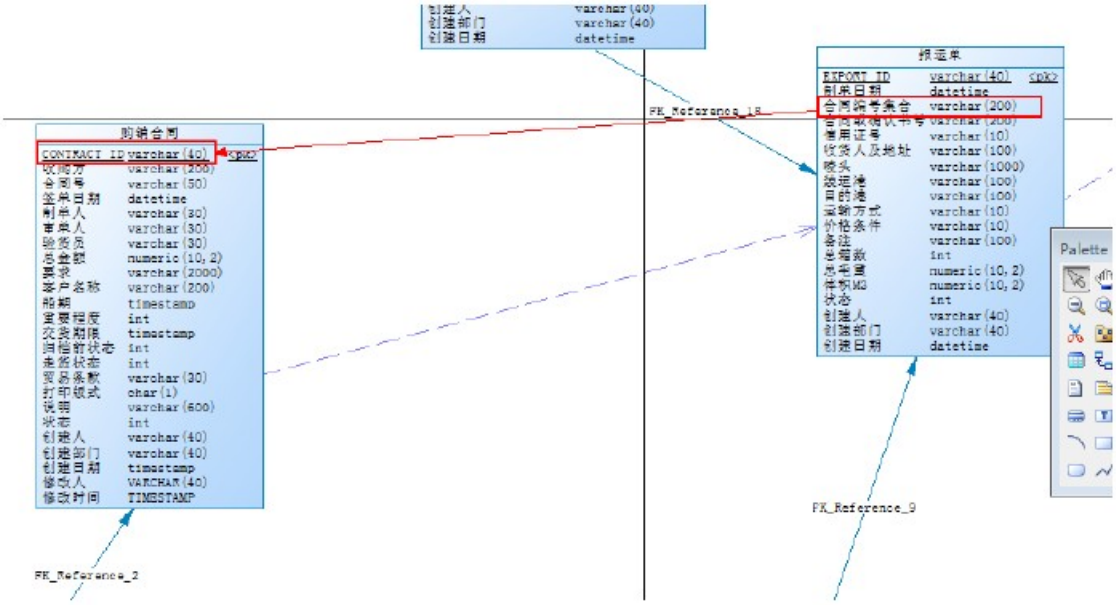
1.打断设计基本概念

就是在传统的一对多关系中，都会在多方加入一个一方的主键作为它的外键，但是在打断设计思想指导下，不会这样实现，它会在一方的表中加入一个冗余字段，用于保存多方

的主键，并且用指定的分隔符进行分隔。

真正的实现原理：就是通过打段字段，实现数据的冗余，从而一样的可以解决一个报运单下有多个购销合同的问题。

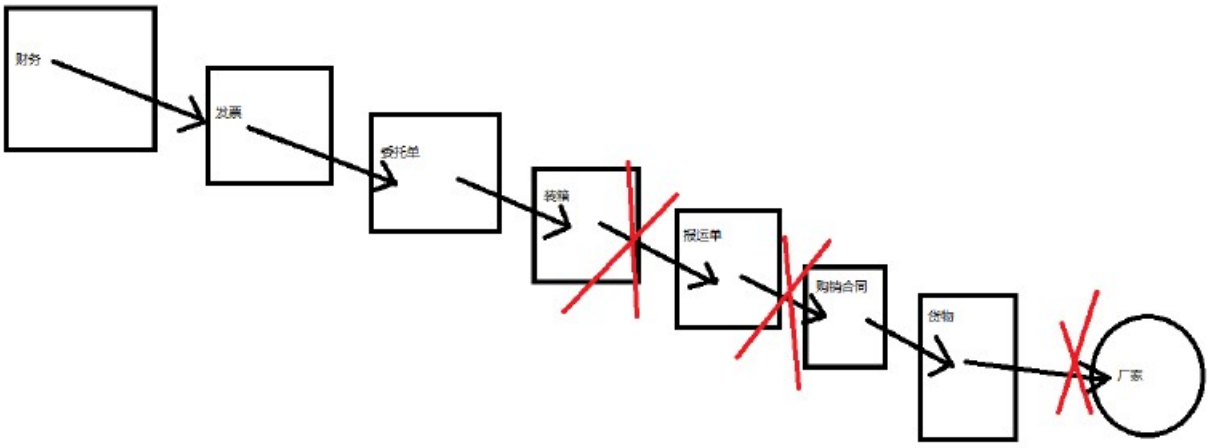
此时的表结构



想知道出口报运单，报运了哪些货物和附件？

2.传统做法（没有使用打断设计）

1.出口报运单对象----------加载出购销合同的集合----------遍历得到每个购销合同----------通过购销合同得到货物----------再通过货物得到附件。



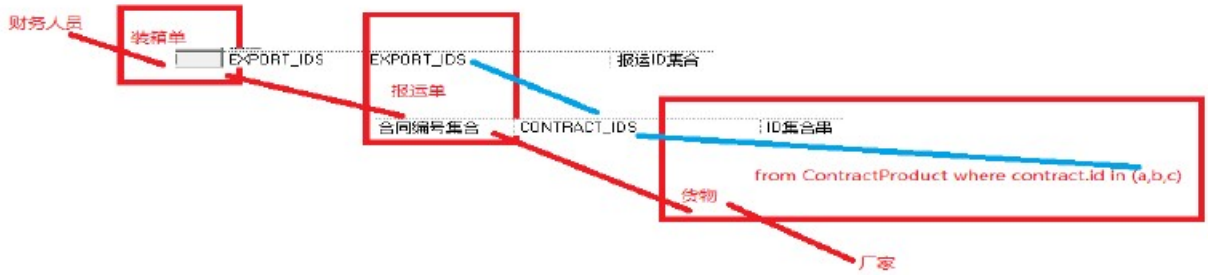
3.打断设计基础上的做法

出口报运单对象----------得到它的一个字段(购销合同 **id** 形成的集合)----------直接到货物表

中进行查询(from ContractProduct cp where cp.contract.id in (购销合同 id 形成的集合))

如果按传统设计方法：
财务想知道这笔款项对应货物/附件是哪个生产厂家生产的。
财务单----------→发票（催款通知单）-------→委托单----------→装箱单----------→报运单
----------→购销合同----------→货物----------→附件----------→生产厂家

采用打断设计实现的优化，如下图



4.打断设计的总结

- 打断设计定义
  它在一方的表中加入一个冗余字段，用于保存多方的主键，并且用指定的分隔符进行分隔。
- 打断设计在什么地方断开？
  打断设计一般在一对一关系，或者一对多关系时可以考虑打断，不建议在多对多关系时进行打断。
- 打断的层级是多少？
  当关联的层级大于 4 层时，就最好考虑打断设计，这样可以实现查询速度的翻倍

5.常见的优化策略

代码优化
算法优化
数据库表结构优化(数据结构的优化)

六．跳跃查询

就是在实现数据查询时，可以跳过中间表而直接进入目标查询对象，从而提高查询速度。
跳跃查询的前提：先进行表结构的打断设计。

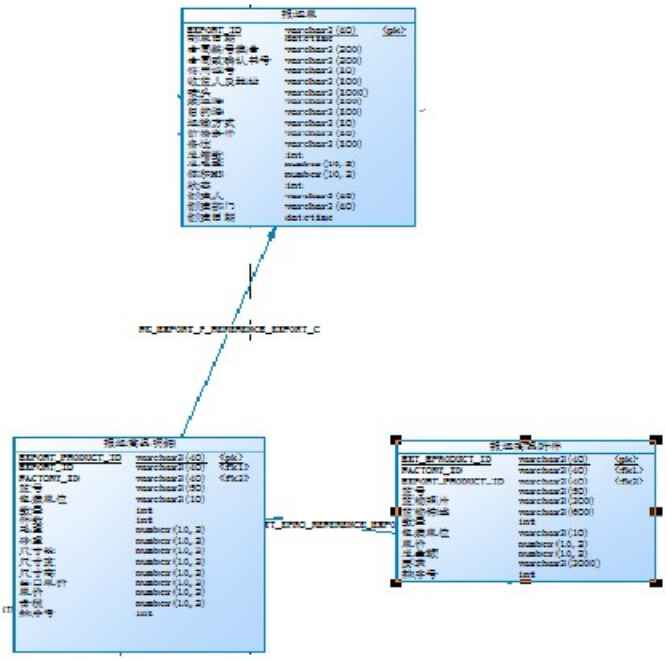
使用打断设计+跳跃查询实现
示例：当我们想知道一个报运单，报运的货物有哪些，可以怎么查询？
此时可以直接通过货物的 contract_id 结合报运单中的 contract_ids 字段一起来实现。

数据库设计上的问题：
早期项目开发交付客户使用后，一切正常。但随着数据量的量变到质变，结果就是系统响应时间成线性增长。系统运行半年后，客户无法承受等待时间。

七．再次优化：数据搬家

表级别的数据冗余。在添加一个出口报运单时，就能得到这个报运单所报运的货物和附件，如何实现报运单下货物和附件有数据，实现原理就是首先找到购销合同对象，再得到购销合同下的货物和附件，针对货物和附件分别进行数据拷贝。

实现手段
手动通过程序代码实现的



八．出口报运模块的实现

1. 加入三个 PO 类及映射文件
2. 编写出口报运模块的 DAO
3. Service 接口及实现类编写及配置

4. Action 编写及配置

5. 注意：修改数据库表的 module_p 的数据

5	2			货运管理	1	0	货运管理	cargo
6	201	2	货运管理	购销合同	2	0	购销合同	cargo/contractAction_list
7	202	2	货运管理	出货表	2	0	出货表	cargo/outProductAction_toedit
8	203	2	货运管理	合同管理	2	0	合同管理	cargo/exportAction_contractList
9	204	2	货运管理	出口报运	2	0	出口报运	cargo/exportAction_list.action
10	205	2	货运管理	装箱管理	2	0	装箱管理	cargo/packingListAction_list
11	206	2	货运管理	委托管理	2	0	委托管理	cargo/shippingOrderAction_list
12	207	2	货运管理	发票管理	2	0	发票管理	cargo/invoiceAction_list
13	208	2	货运管理	财务管理	2	0	财务管理	cargo/financeAction_list

将你的表中数据，调成与上面相同

九．作业

1.完成读程的工作，并实现购销合同的打印功能
2.完成出口报运单的生成
3.完成报运单的查询&删除&查询详情，更新操作有难度（可以暂时不实现）
4.实现细粒度权限控制之管理本部门及下属部门