报表模式化开发手册

（V1.0, for NC-V3）

朱俊彬、赖宏伟、李媛媛

NC-UAP

目　　录

第一章　总体介绍 2

1. 开发概述 2
1.1 技术型报表与业务型报表 2
1.2 一般报表开发存在的问题 2
1.3 模式化报表开发思路 2
2. 应用模型 3
2.1 数据模型与展现模型 3
2.2 数据字典 4
2.3 业务系统集成 5

第二章　低开发难度报表 6

1. 概述 6
2. 查询类报表 8
2.1 明细报表 8
2.2 汇总报表 17
2.3 列结构依赖于查询条件的报表（动态SQL） 23
3. 交叉类报表 30
3.1 列向拼接查询型报表（复合查询） 31
3.2 列向分支统计型报表（CASE-WHEN） 36
3.3 动态行列交叉型报表（旋转交叉） 40
4. 投影类报表 45
4.1 单元格依赖于行列条件的报表（投影交叉） 45
4.2 半录入半嵌入型报表（合并查询） 51
5. 算法类报表 55
5.1 数据加工预备知识 55
5.2 非投影类占比报表（普通占比） 60
5.3 投影类占比报表（投影占比） 65
5.4 程序送数型报表 69
6. 高级应用 73
6.1 支持穿透的非投影交叉类报表（普通穿透） 73
6.2 支持穿透的投影交叉类报表（投影穿透） 79
6.3 支持主从连动的报表 84
6.4 支持统计图表的报表 88

第三章　高开发难度报表 92

1. 基于行业报表工具开发的报表 92
1.1 标准报表 92
1.2 分块填充报表 104
2. 基于CELL REPORT工具开发的报表 111
2.1 CELL REPORT工具介绍 111
2.2 CELL REPORT报表 113

附录 124

1．模式一览表 124
2．查询引擎FAQ之参数篇 125

第一章　总体介绍

1. 开发概述

1.1 技术型报表与业务型报表

我们将报表应用划分为两种类型，一类应用的侧重点在于出报表所蕴涵的技术，开发者（程序员或实施人员）通常凭借一定的数据库和编程知识，基于库里的业务数据，做出符合用户需求的报表展现和输出，此类应用称为技术型报表；另一类应用则着眼于整个报表生命周期中各环节的管理（包括出表、上报、审核、汇总、发布等流程），即把做报表的行为当成一项业务，对直接开发者的主要要求在于业务熟悉程度而非技术，此类应用称为业务型报表。

前者通常由专业程序员、二次开发人员或技术型实施顾问承担开发工作，早期工具如报表模板、自定义查询等均面向此类应用，IUFO 则是后者的代表性产品。本文档作为集团平台技术部对外发布的开发手册，主要针对对象为技术型报表。

1.2 一般报表开发存在的问题

报表属于 ERP 项目中的关键应用，通常被用户的高层领导用于企业业绩评估、辅助决策或对外发布。据统计，在很多交付压力较大的项目当中，报表开发占有相当的比重。就目前开发现状而言，影响报表快速交付的主要问题有以下几点：

1、 报表开发本身的复杂性高。中国式报表由于政策以及长期人工填报的原因具有极强的个性化特点，根据业务逻辑进行数据查询、数据组织、表样展现和界面处理的每个步骤都耗费了很高的开发代价；
2、 报表开发的复用性低。开发人员投入巨大努力完成第一批报表后，发现第二批报表依然要从头做起，或者另一名开发人员遇到类似于第一批报表的需求，无法借鉴（甚至根本不知道）前人的开发成果；
3、 报表开发对专业技能要求偏高。很多二次开发人员虽然不是专门搞技术的，但通常具有一定的开发基础，而目前的二次开发平台限制了他们发挥应有的作用。

1.3 模式化报表开发思路

所谓模式化开发，包含两方面的工作：专职人员将有代表性的应用进行归纳抽象，提取出共性的部分形成代码框架和方案，并以手册的形式对外发布；报表开发人员拿到项目需求后，从手册上找到匹配的模式，通过编写一些低技术含量的子类代码或进行一些机械性的工具操作，就能够快速地完成开发工作。以往大部分的重复性劳动已经在模式的基类或者缺省实现

中处理，当开发人员需要体现个性化的时候，则可以通过自己对接口的特殊实现完成。

随着模式的逐步丰富，开发人员开发各类模式报表的工作量都会明显降低，同时也实现了代码复用和知识共享，另一方面，由于开发者的工作趋于简单和机械，他们的角色就可以由低专业技能人员所取代。从这个角度上说，模式化是解决上节所述三个问题从而真正实现报表快速开发的有效方案。

本手册介绍了两大类 NC 报表开发模式，一类是基于查询引擎的低开发难度报表，其编码量小，操作简单，适合于开发数据和格式比较规律的报表；另一类是基于行业报表工具的高开发难度报表，主要工作是编写子类代码，自由度高，适合于开发复杂报表。

2. 应用模型

2.1 数据模型与展现模型

一般来说，报表是由数据和格式组成的整体。报表的数据模型通常包含以下信息：

1、 数据结构；
2、 待定参数描述；
3、 取数规则描述；
4、 数据加工算法描述；
5、 多个数据集合之间的数据关联描述；

可作为数据载体的常用 NC 数据结构包括：

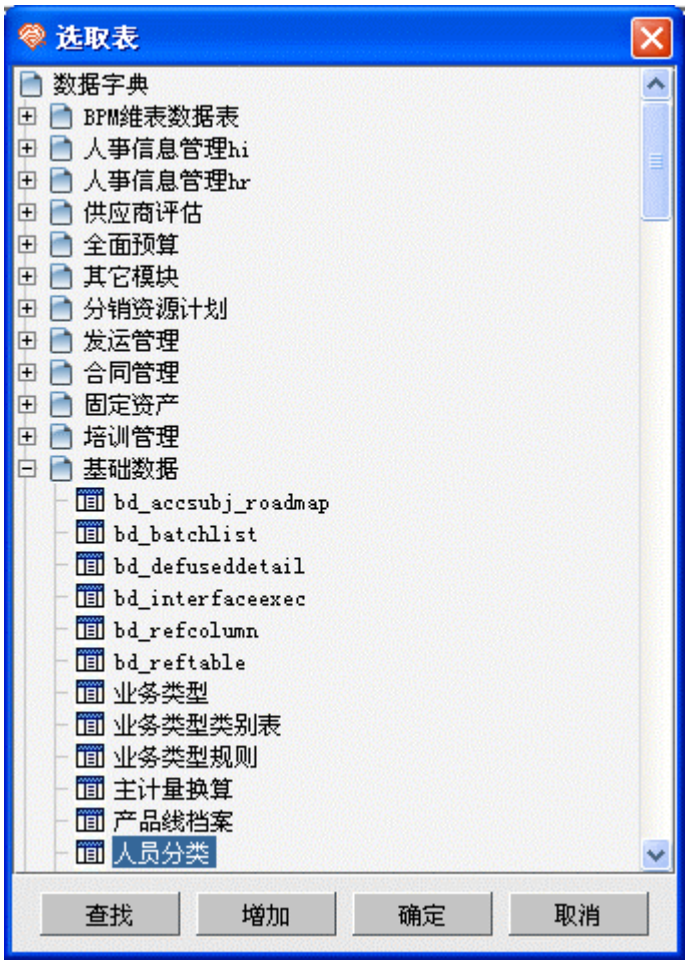
1、 nc.vo.pub.CircularlyAccessibleValueObject——循环访问 VO，是由属性和取值成对组成的集合，提供了根据属性获得/设置值的方法，是单据和报表模板采用的主要结构；
2、 nc.vo.pub.ValueObject——普通 VO，包含多个属性及每个属性的 getter 和 setter 方法，是业务组主要数据结构的父类；
3、 com.borland.dx.dataset.StorageDataSet——存储数据集，由元数据和内容数据两部分组成，元数据（Column[]）描述了各列的信息，内容数据描述了一个二维的数据集合，当游标指向内容数据的某行时，可以获得该行指定列的数据，用于查询引擎；
4、 nc.vo.pub.rs.MemoryResultSet——内存结果集，同样由元数据和内容数据组成，其中元数据（MemoryResultSetMetaData）描述各列信息，内容数据（ArrayList）描述了一个二维数据集合，可以直接访问其任何元素，目前被一些业务算法使用；
5、 Object[][]，Vector，ArrayList——这些都是 JDK 的常用结构，均可存储二维数据。

报表的展现模型通常包含以下信息：

1、 控件（核心控件是表格）属性和布局；
2、 控件显示内容与显示规则（绑定数据、参数、行列格式、界面资源等）；
3、 控件连动规则；
4、 数据处理规则（定位、排序、过滤、统计、公式、交叉、钻取等）；
5、 动态扩展规则；
6、 输入输出控制（待定参数设置，打印，导出等）

2.2 数据字典

NC 数据字典提供了统一的数据建模平台，支持对各种数据库对象进行管理，维护这些物理对象的逻辑属性，并向外系统提供访问数据库逻辑信息的接口。以查询引擎为例，查询对象的定义是基于数据字典进行的，而这份数据字典来自查询对象所指定的数据源。由于多数报表的列与业务数据库的字段存在某种对应关系，因此数据字典的存在有助于提高查询定义的直观性和易用性。



NC 数据字典通常在安装产品时候由系统生成，如果想自己为某些数据库对象生成字典，可以采用数据字典管理界面提供的导入方法。导入方式包括三种（详细操作可参见《数据字典导入说明》）：

1、 解析 PDM 文件（xml 格式）；
2、 解析 NC 标准建库脚本；
3、 提取数据库元数据；

如果需要给非 NC 数据库（比如用户采用的第三方数据库）生成数据字典，可以利用 PowerDesigner 提供的逆向工程功能将库里的物理表生成 PDM 文件，修改其中表和字段的中文名称，然后以 xml 文件格式存储，再使用上述第一种方法导入。

2.3 业务系统集成

对于直接使用查询引擎开发出的报表，可以使用以下两种方式挂接到业务组的功能节点。强烈建议把准备挂接节点的查询对象和格式对象的编码设为与业务系统相关的有意义字符串，以避免不同业务系统预置对象互相覆盖的危险性。

报表管理节点挂接法：

1、 注册功能节点，类名为 nc.ui.pub.querymodel.QueryMainUI_N；
2、 下挂一个参数，参数名为 folderId，参数值为客户化下查询引擎管理中相关报表目录的 ID（可从数据库表 pub_formatmodeldef 的 id 字段查出）。

独立报表节点挂接法：

1、 注册功能节点，类名为：nc.ui.pub.querymodel.QueryNodeUI：
2、 下挂两个参数：pkQryNode——界面模型对应的编码（可从查询引擎管理界面读取），dsName——查询定义所在数据源。

对于高开发难度模式的报表，则与普通功能节点一样挂接 ToftPanel 的子类即可。

第二章 低开发难度报表

1. 概述

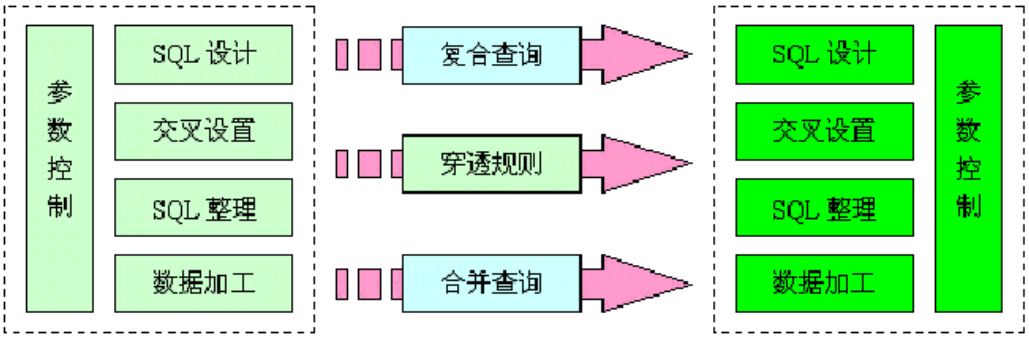
本手册列举的低开发难度报表模式统一由查询引擎（V3 版本）提供解决方案。查询引擎是一个面向高级实施人员和普通开发人员的查询建模产品，延续并完善了自定义查询体系的技术路线，在全面支持复杂查询的设计和个性化的报表展现的同时，大幅度降低了报表开发人员的编码工作量和专业技能要求。

我们再来看一下查询引擎的适用性和不适用性。查询引擎长期作为一个查询工具发展，因此尤其适用于与数据库查询分析或数据挖掘关系密切的报表开发，同时也支持程序员调用服务器端代码对数据进行业务处理。另一方面，自定义查询体系与模板体系存在一个很大的不同，模板体系在发布产品之前已经由程序员录入了大量初始化数据，因此不同用户能够分配到不同的查询模板、报表模板和打印模板，而自定义查询体系是完全自定义的实时开发，没有模板的预制数据，因此在一个时刻只支持一种参数控制样式、一种报表界面样式和一种打印样式（或直接打印），但这些样式均可在设计态修改。在展现上，由于查询引擎使用的是 SWING 的表格控件，因此对一些 EXCEL 风格的需求支持不够，这个薄弱环节将在 V31 得到加强。总体来说，V3 版本的查询引擎暂不适用于以下两类报表开发：

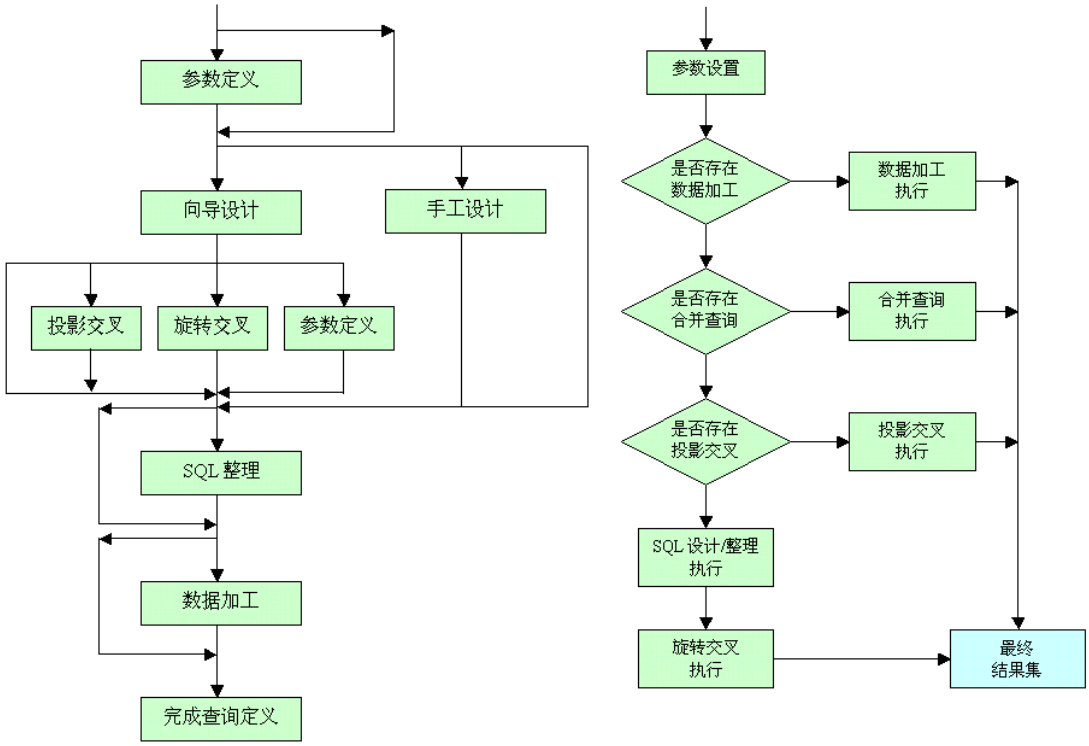
1、 有多模板分配需求的报表，或对打印有高度要求的报表；

2、 存在表体合并单元格的报表。

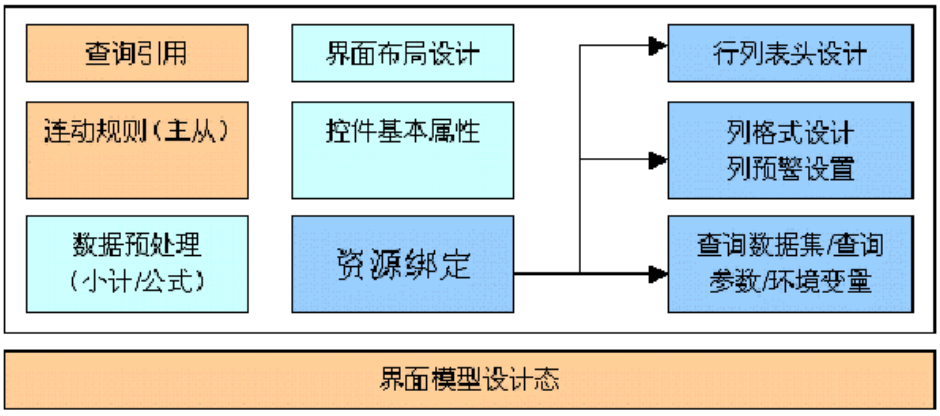
查询引擎的查询模型（数据模型）设计如下：

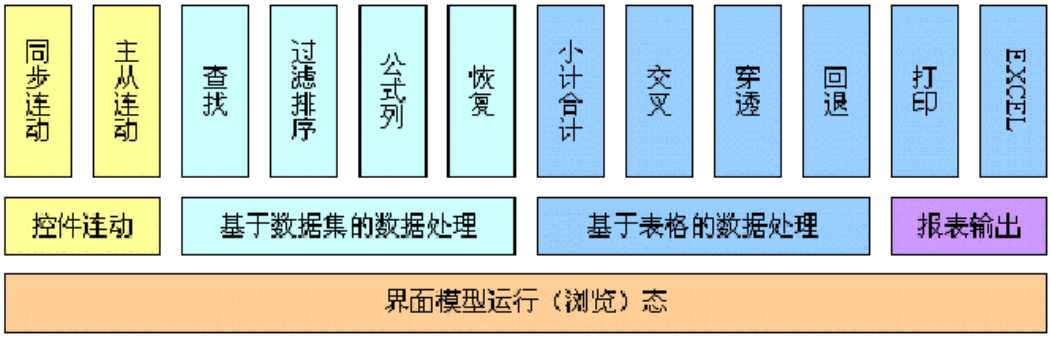


查询模型流程如下所示（设计态/运行态）：

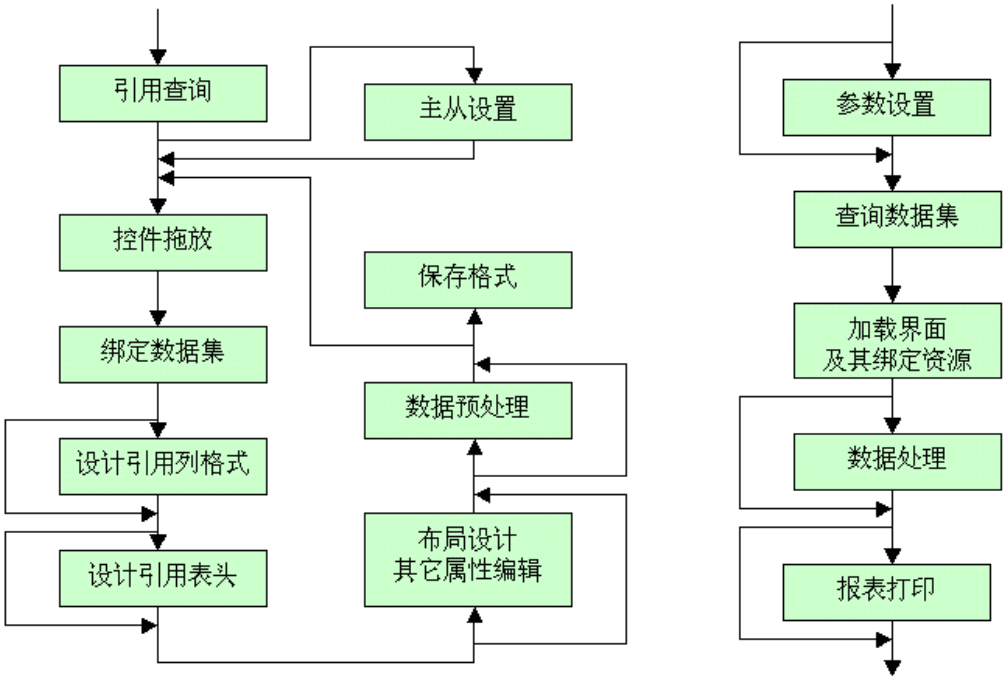


界面模型（展现模型）设计如下：





界面模型流程如下所示：



本章共介绍了五大类报表的十五种开发模式。尽管这些模式之间存在相当差异，但是针对以前我们发现的一些用例杂乱导致的培训效果问题，本章的所有范例集中面向两个业务应用（单位费用统计和部门人事统计），即用不同方案解决相似问题，以便于读者从中领会和比较这些模式的原理、特点和适用性。希望这种做法不会给读者造成一个误导：查询引擎只能解决这两类报表的应用。

2. 查询类报表

查询类报表指通过一个或多个 SQL 查询结果形成的二维数据展现出来的报表，由于 SQL 的语法特点，此类报表通常具有固定的列结构和动态的行结构，但模式 2.3 也探讨了一种根据参数动态改变查询列结构的方案。

2.1 明细报表

【概述】

明细报表用于展现最基本维度上的数据，即最细粒度的数据。通常明细总是相对于汇总而言的，如果我们把某商品每个月的总销售额列表视为汇总数据，那么该商品每天的详细销售情况就可视为明细数据；如果我们把某部门当年的总薪资情况做成汇总表，那么该部门每位员工当年的薪资情况就可以做成明细表。查询类报表中的明细表通常是指一些不含聚合函数的 SQL 语句直接能够查询出的数据形成的报表。

【应用场景】

开发人员需要通过多表联查 SQL 或者复合查询获得表体数据，同时为报表展现提供待定条件设置、栏位设置、排序、过滤、定位、小计合计及输出等功能。当明细表与汇总表一起出现时，用户通常还会有从汇总数据联查明细数据的需求（又称穿透或钻取），此类应用将在模式 6.1 中介绍。

【适用性和不适用性】

适用于：

- 明细数据的取数规则可用 SQL 描述；

不适用于：

- 通过行列条件统计单元取值的明细表。此类报表应遵循模式 4.1 解决；
- 通过程序算法构造明细数据的报表。此类报表应遵循算法类报表模式解决；

【解决方案】

取数由向导式 SQL 设计描述，查询条件由参数控制机制解决，栏目在格式设计态设定，排序、过滤、定位、小计合计、输出等功能均内置于报表浏览态。

【开发步骤】

第一步：在查询引擎管理中建立查询对象 Q1，做向导式 SQL 设计（通常不含聚合函数 sum、avg、count、max、min）。对于一个 SQL 无法描述的查询，比如指定单位在指定期间内不同科目的发生额，可以利用复合查询（基于查询对象的查询，可参考模式 3.1）进行描述。Q1 中待定的信息由参数定义描述，普通参数可以在向导式设计的筛选条件处引用，替换型参数可以在向导式和手工式设计的任意位置引用（参见 FAQ 文档）；

第二步：创建格式对象 F1，引用查询对象 Q1 并嵌入表格，做相关的栏位、列格式和列表头设计。表头表尾可放置绑定参数的控件；

第三步：浏览 F1（或挂功能节点），设置参数，检查数据与格式是否正确；

【FAQ】

1、如何充分发挥查询模型中参数的功能？
答：参见附录提供的《查询引擎 FAQ 之参数篇》。

2、格式设计中的 16 种控件各有何种用途？
答：表格、图表是核心控件，用于绑定数据集；面板、拆分窗格（分割面板）、多页签作为界面容器；标签、文本框、下拉框（组合框）、参照、文本域、复选框、单选框是表头表尾控件，需要放在容器上面，用于绑定参数；下拉框、列表绑定枚举型参数，参照绑定参照型参数；主子表是已经不推荐使用的连动展现控件；按钮、树暂无实际用处。

3、如何控制报表的输出格式？
答：打印输出利用格式设计中的打印设置功能控制，EXCEL 输出由界面表格的样式控制。

【范例】

单位费用明细表——各单位在指定期间范围内的费用明细情况。报表中要求包括公司、期间、制单日期、科目、分录摘要、发生额等信息，用户可以根据年度和期间范围进行查询。

公司	期间	制单日期	科目	摘要	借发生额	贷发生额

1、 查询引擎管理主界面创建查询对象，编码为 gldetail，名称为费用明细；

2、进行参数定义，增加三个字符型参数：年，起始期间，终止期间；

3、进行 SQL 向导设计：从数据字典添加凭证表（gl_voucher）、凭证分录（gl_detail）、科目表（bd_accsubj）和公司目录（bd_corp），指定表间连接关系、查询字段、筛选条件（先定义确定条件）和排序字段；


查询定义
选择表
连接条件
选择字段
筛选条件
排序字段
交叉属性
表物理名
表显示名
表类型
1 gl_voucher 凭证表 数据表
2 gl_detail 凭证分录 数据表
3 bd_accsubj 科目表 数据表
4 bd_corp 公司目录 数据表
增加表
临时表
删除表
上移
下移
上一步
下一步
确定
取消


查询定义
选择表
连接条件
选择字段
筛选条件
排序字段
交叉属性
典型模式
高级模式
JOIN 部分
ON 部分
连接表1
连接方式
连接表2
连接字段1
比较符
连接字段2
1 凭证表 内连接 公司目录 公司编码 = 公司主键
2 凭证表 内连接 凭证分录 凭证主键 = 凭证主键
3 凭证分录 内连接 科目表 科目主键 = 科目主键
增加
删除
上移
下移
互换
上一步
下一步
确定
取消

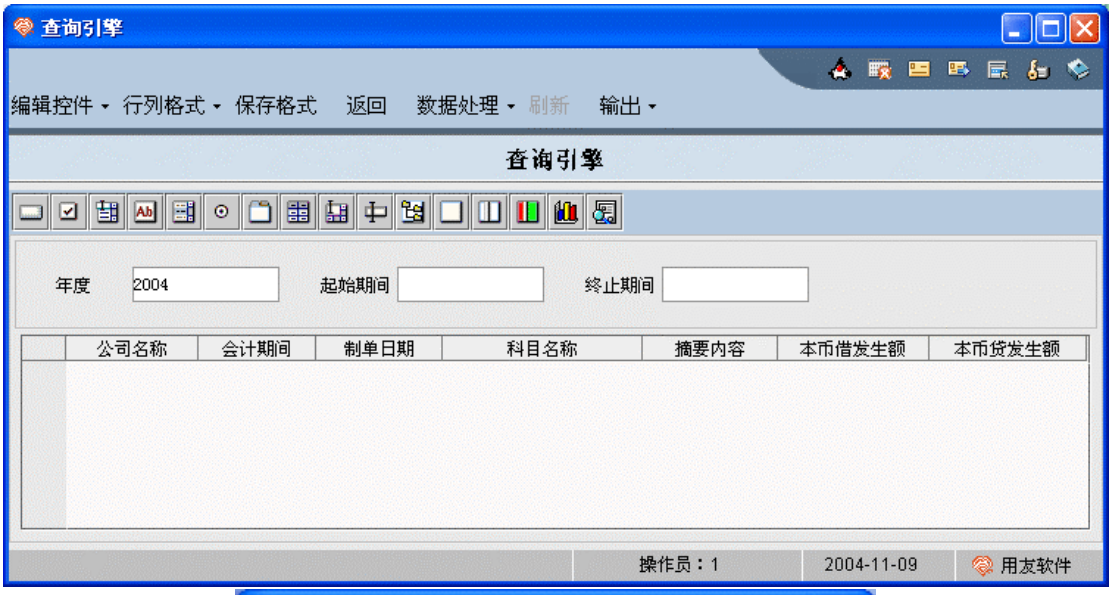

查询定义
选择表
连接条件
选择字段
筛选条件
排序字段
交叉属性
字段表达式
显示名
别名
1 bd_corp.unitname 公司名称 unitname
2 gl_voucher.period 会计期间 period
3 gl_voucher.prepareddate 制单日期 prepareddate
4 bd_accsubj.subjname 科目名称 subjname
5 gl_detail.explanation 摘要内容 explanation
6 gl_detail.localdebitamount 本币借发生额 localdebitamount
7 gl_detail.localcreditamount 本币贷发生额 localcreditamount
增加
修改
删除
上移
下移
上一步
下一步
确定
取消

4、在筛选条件页签增加待定条件，在右操作数框中按 F12 引用参数；

5、创建格式对象，编码为 gldetail，名称为“费用明细表”；

6、引用查询“费用明细”；

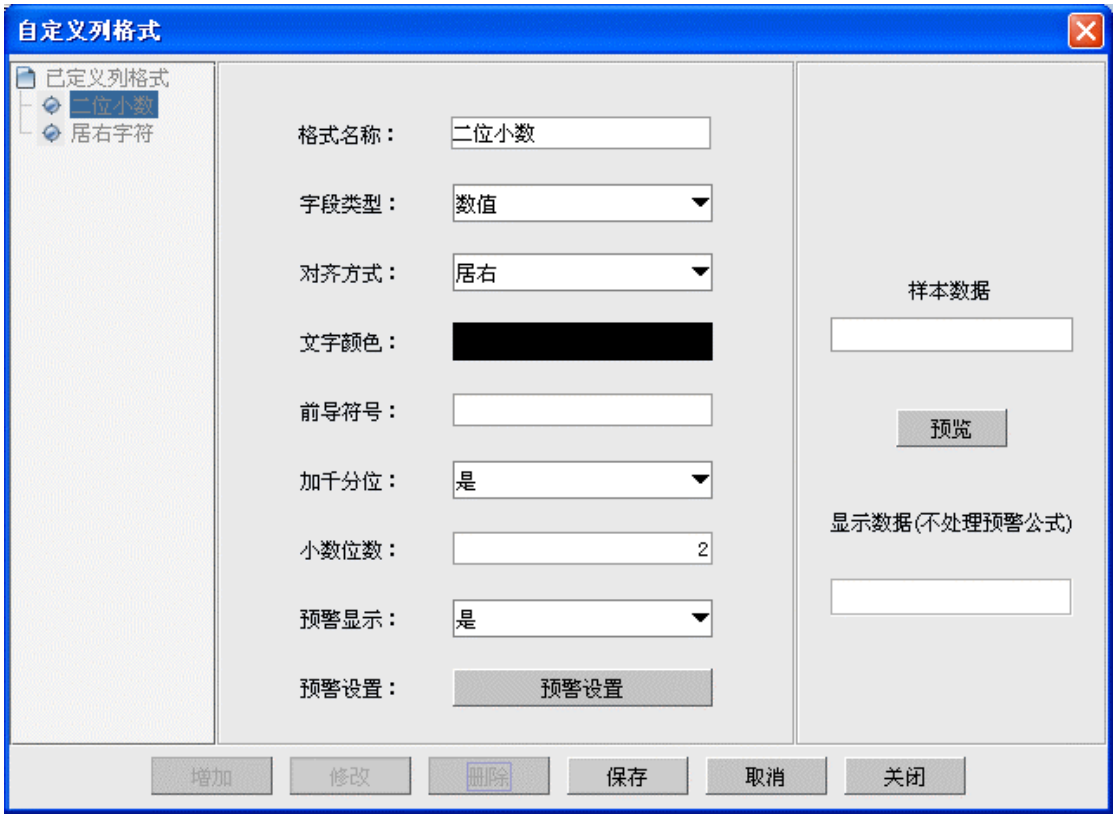
7、进行格式设计：添加表格和面板控件，表格停靠于中部，双击控件设置属性，绑定数据集费用明细，面板停靠于北部（作为表头容器），采用流式布局（FlowLayout），在面板上放三个文本框控件，分别绑定费用明细的三个参数；


查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
查询引擎
年度
2004
起始期间
终止期间
公司名称
会计期间
制单日期
科目名称
摘要内容
本币借发生额
本币贷发生额
操作员：1
2004-11-09
用友软件


属性编辑器
控件:
表0
属性
取值
绑定数据集
费用明细
背景色
255, 255, 255
前景色
0, 0, 0
字体
宋体
12
A
停靠属性
中
首选大小
Width:
945
Height:
367
表头ID
列格式
公司名称
提示文本


属性编辑器
控件:
面板1
属性
取值
背景色
233, 233, 233
前景色
0, 0, 0
字体
Dialog
12
A
停靠属性
北
布局
FlowLayout
首选大小
Width:
431
Height:
60
提示文本

8、（可选）定义一种列格式，用于显示两位小数，在表格属性框的列格式分栏中由两个金额列引用此列格式；



9、浏览费用明细表（先设置参数）；

	公司名称	会计期间	制单日期	科目名称	摘要内容	本币借发生额	本币贷发生额
1	无锡分公司	01	2004-01-02	长江工程公司	工程款	0.00	800.00
2	无锡分公司	01	2004-01-02	长江工程公司	付工程款	0.00	50.00
3	无锡分公司	01	2004-01-02	长江工程公司	内部转帐	8,000.00	0.00
4	无锡分公司	01	2004-01-02	长江工程公司	代付款	30.00	0.00
5	无锡分公司	01	2004-01-02	长江集团总部	工程款	800.00	0.00
6	无锡分公司	01	2004-01-02	长江集团总部	代收款	0.00	3,000.00
7	无锡分公司	01	2004-01-02	长江集团总部	广告费	750.00	0.00
8	无锡分公司	01	2004-01-02	长江政务公司	付工程款	50.00	0.00
9	无锡分公司	01	2004-01-02	长江政务公司	委托收款	80.00	0.00
10	无锡分公司	01	2004-01-02	东莞电子制造厂	委托收款	0.00	80.00
11	无锡分公司	01	2004-01-02	东莞电子制造厂	代收款	0.00	5,000.00
12	无锡分公司	01	2004-01-02	东莞电子制造厂	内部转帐	0.00	8,000.00

2.2 汇总报表

【概述】

汇总数据通常与明细数据相对，是指在明细数据基础上指定若干个有分类意义的字符型列进行分组，并对另外若干个有统计意义的数值型列进行求和（或求平均、最值、计数等）获得的数据。展现汇总数据的报表称为汇总报表。

【应用场景】

我们再细分为两种应用：

1. 静态汇总设置：
   查询数据的 SQL 中指定了分组列和汇总列，此设置在报表浏览状态不再改变。要求提供待定条件设置、栏位设置、排序、过滤、定位、小计合计及输出等功能。
2. 动态汇总设置：
   事先对数据设定某种缺省的汇总设置，此设置在浏览态可做更改，并重新计算汇总数据。同时具备上述功能需求。

【适用性和不适用性】

适用于：

- 汇总数据的取数规则可用 SQL 描述；

不适用于：

- 通过行列条件统计单元取值的汇总表。此类报表应遵循模式 4.1 解决；
- 通过程序算法构造汇总数据的报表。此类报表应遵循算法类报表模式解决；

【解决方案】

取数由向导式 SQL 设计描述，查询条件由参数控制机制解决，栏目在格式设计态设定，动态汇总由旋转交叉机制解决，排序、过滤、定位、小计、输出等功能均内置于报表浏览态。

【开发步骤】

1、静态汇总设置

第一步：在查询引擎管理中建立查询对象 Q2，做向导式 SQL 设计，定义带有聚合函数的 SQL 语句。单个 SQL 无法描述的查询利用复合查询描述。在 Q2 的查询模型中创建参数和引用参数，用来描述待定的信息；

第二步：创建格式对象 F2，引用查询对象 Q2 并嵌入表格，做相关的栏位、列格式和列表头设计。表头表尾可放置绑定参数的控件；

第三步：浏览 F2（或挂功能节点），设置参数，检查数据与格式是否正确；

2、动态汇总设置

第一步：同上述第一步（但也可以不设置聚合函数，而是在旋转交叉设置中把缺省的分组列放到行列表中，把缺省的汇总列放到值列表中）；

第二步：同上述第二步，因交叉表的列结构不能在设计态确定，因此只能指定非汇总列的列格式等信息；

第三步：同上述第三步，需要改变汇总设置的时候，利用表头右键菜单的交叉功能把新的汇总列移到交叉行列表，确定即可；（关于交叉的详细说明参见下一节）

【FAQ】

1、 行 SQL 向导设计时如何指定 group by 字段？
答：无需指定，系统自动把不含聚合函数的查询字段作为 group by 字段。

2、 行 SQL 手工设计时为何 order by 字段无效？
答：需要显式指定排序字段的排序方式（升序——asc，降序——desc）。

【范例】

单位费用汇总表——各单位在指定期间范围内的费用汇总情况。报表中要求包括公司、期间、科目、汇总发生额等信息，用户可以根据年度和期间范围进行查询。

公司	期间	科目	借发生额	贷发生额

1、 创建查询对象，编码为 glsum，名称为费用汇总；

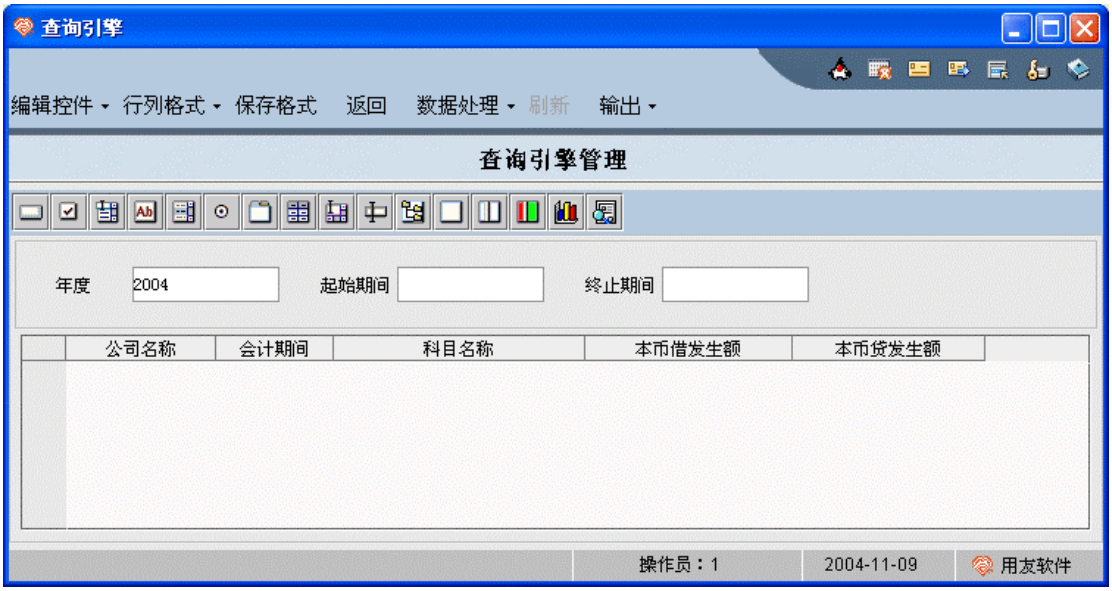
2、进行参数定义，增加三个字符型参数：年，起始期间，终止期间；

3、进行 SQL 向导设计：从数据字典添加凭证表、凭证分录、科目表和公司目录，指定表间连接关系、查询字段、筛选条件和排序字段（除查询字段外均与模式 2.1 的范例相同）；

4、在筛选条件页签增加待定条件，在右操作数框中按 F12 引用参数；

5、创建格式对象，编码为 glsum，名称为“费用汇总表”。引用查询“费用汇总”；

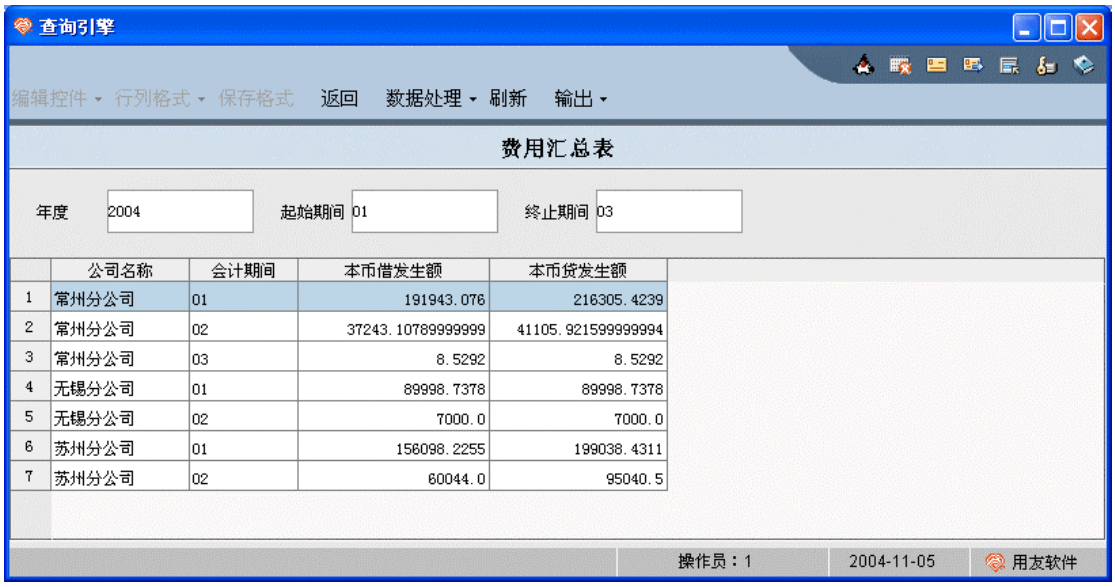
6、进行格式设计：添加表格和面板控件，表格停靠于中部，绑定数据集费用汇总，面板停靠于北部（作为表头容器），采用流式布局，在面板上放三个文本框控件，分别绑定费用汇总的三个参数；



7、浏览费用汇总表（先设置参数）；

	公司名称	会计期间	科目名称	本币借发生额	本币贷发生额
1	无锡分公司	01	长江工程公司	8030.0000	4850.0000
2	无锡分公司	01	长江集团总部	17523.4245	11708.0000
3	无锡分公司	01	长江政务公司	130.0000	8.0000
4	无锡分公司	01	贷款	74.0000	331.7500
5	无锡分公司	01	定期存款	0.0000	3.0000
6	无锡分公司	01	东莞电子制造厂	14568.6500	22696.8400
7	无锡分公司	01	工行存款	14655.5000	44943.8145
8	无锡分公司	01	广州结算中心	574.7500	139.0000
9	无锡分公司	01	金融机构借款	11446.5800	67.5000
10	无锡分公司	01	利息费用	2275.8333	0.0000
11	无锡分公司	01	内部贷款利息	0.0000	1000.0000
12	无锡分公司	01	上海结算中心	0.0000	2870.8333
13	无锡分公司	01	外部贷款	20000.0000	0.0000
14	无锡分公司	01	外部单位	720.0000	1380.0000
15	无锡分公司	02	长江集团总部	7000.0000	0.0000
16	无锡分公司	02	工行存款	0.0000	7000.0000
17	苏州分公司	01	本年利润	0.0000	181.8144
18	苏州分公司	01	财务费用	1000.0000	1000.0000
19	苏州分公司	01	待摊费用	1.2000	1000.0000
20	苏州分公司	01	固定资产	34530.0000	29.6100

8、当前汇总设置是对公司、期间、科目分组统计发生额，如果需要动态改变汇总依据，则可以使用表头右键菜单中的交叉功能。比如我们需要按照公司、期间重新汇总，则把公司和期间移至交叉行列表，发生额移至交叉值列表，确定即可；



9、如果我们把科目移至交叉行，期间移至交叉列，发生额移至交叉值，那么就实现了带有数据旋转的动态汇总，即展现各科目在不同期间下汇总出来的发生额。关于旋转交叉的详细应用可参看模式 3.3；

	科目名称	本币借发生额			本币贷发生额		
		01	02	03	01	02	03
1	上海结算中心	0.0			2870.8333		
2	东莞电子制造厂	14568.65			22696.84		
3	主营业务成本	23997.328			566.3313		
4	主营业务收入	951.8603	0.0		951.8602	196.25	
5	其他业务收入	40.0	0.0		40.0	40.0	
6	其他应收款	15834.1			16034.08		
7	其他资本公积	2500.0	0.0		10000.0	1000.0	
8	内部往来	20006.603000000003	20002.0		10022.131	0.0	
9	内部贷款利息	0.0			1000.0		
10	内部银行存款	34404.84	0.0		32092.074500000002	2000.0	
11	利息费用	2275.8333			0.0		
12	原材料	5.4			0.0		
13	固定资产	64052.441	27009.879999999997		2534.61	10003.1	
14	固定资产减值准备	4.58	0.0		2804.58	3900.0	
15	固定资产清理	17.3936	9852.6639		0.0	0.0	
16	固定资产购入	24.891	0.0		24522.441	16696.5879	
17	基本生产成本	9.6552		8.5292	2002.44		0.0
18	外部单位	720.0			1380.0		
19	外部应收	15086.222	10014.0		87257.094	60000.0	

2.3 列结构依赖于查询条件的报表（动态 SQL）

【概述】

我们前边提到过，SQL 查询的结果具有固定的列结构和动态的行结构，前两个模式中虽然都引入了参数来控制动态信息，但也只用于待定的筛选条件，即影响报表的行结构。然而有一类报表因查询条件的不同而同时具有动态的列结构和行结构，比如在供应链的一些单据查询报表中，如果用户在查询条件中勾选了联查订单或到货单，那么报表会在静态的列结构（可能有 10 列）后面追加一些动态的列或列分组，用于显示联查单据的明细信息（此时报表可能就变成了 20 列）。

【应用场景】

开发人员需要把静态的查询 SQL 同用户对查询条件设置的取值结合起来，构造出几种不同的 SQL，分别对应于用户希望看到的各种报表展现结果。这些 SQL 不仅仅存在筛选条件的差异，也可能存在查询字段、查询表、分组字段、排序字段的差异。由于本模式采用的技术方案需要通过代码片段来描述查询 SQL 究竟被如何矫正，因此本模式仅建议熟悉 JAVA 编程的报表开发人员使用。

【适用性和不适用性】

适用于：

- 查询结果可由规范的静态 SQL 和动态参数取值联合决定；

不适用于：

- 查询必须通过难以解析的复杂手工 SQL 才能描述；
- 要求对动态增加出来的列结构进行列格式设计的报表（因为这些列在设计态是未知的）；

【解决方案】

取数由向导式 SQL 设计描述，查询条件由参数控制机制解决，如何根据参数取值调整静态 SQL 则由 SQL 整理中的代码片段来描述。

SQL 整理中的常用数据结构和接口方法如下所述：

1、 查询基本定义类 QueryBaseDef，所提供方法包括：

FromTableVO[] getFromTables(); //获得查询字段定义
SelectFldVO[] getSelectFlds(); //获得查询字段定义
JoinCondVO[] getJoinConds(); //获得连接条件定义
WhereCondVO[] getWhereConds(); //获得筛选条件定义
GroupbyFldVO[] getGroupbyFlds(); //获得分组字段定义
OrderbyFldVO[] getOrderbyFlds(); //获得排序字段定义
以及相应 setter（设置）方法。

2、 查询表定义类 FromTableVO，所提供方法包括：

String getTablecode(); //获得表物理名
String getTabledisname(); //获得表显示名
String getTablealias(); //获得表别名
void setTablecode(String); //设置表物理名
void setTabledisname(String); //设置表显示名
void setTablealias(String); //设置表别名

3、 查询字段定义类 SelectFldVO，所提供方法包括：

String getExpression(); //获得字段表达式
String getFldname(); //获得字段显示名
String getFldalias(); //获得字段别名
void setExpression (String); //设置字段表达式
void setFldname (String); //设置字段显示名
void setFldalias (String); //设置字段别名

4、 连接条件定义类 JoinCondVO，所提供方法包括：

String getExpression0(); //获得筛选条件表达式
void setExpression0(String); //设置筛选条件表达式

5、 筛选条件定义类 WhereCondVO，所提供方法包括：

String getExpression0(); //获得筛选条件表达式
void setExpression0(String); //设置筛选条件表达式

6、 分组字段定义类 GroupbyFldVO，所提供方法包括：

String getExpression(); //获得分组字段
void setExpression(String); //设置分组字段

7、 排序字段定义类 OrderbyFldVO，所提供方法包括：

String getExpression(); //获得排序字段
UFBoolean getAsc(); //获得升序标志
void setExpression(String); //设置排序字段

void setAsc(UFBoolean); //设置升序标志

8、 SQL 整理中获得静态查询基本定义的方法为

QueryBaseDef getQueryBaseDef();

整理代码的核心目标就是根据参数取值来调整这个 QueryBaseDef 的内部结构。

【开发步骤】

第一步：在查询引擎管理中建立查询对象 Q3，做向导式 SQL 设计，描述静态的 SQL 语句。在 Q3 的查询模型中创建参数和引用参数，用来描述待定的信息；

第二步：在查询模型中编写 SQL 整理代码，这段代码采用纯 JAVA 语法，目的是根据用户的参数设置取值调整静态的查询基本定义 QueryBaseDef，并把调整后的 QueryBaseDef 所生成的 SQL 发到数据库去执行。整理界面的向导树内置了常用的整理代码向导，可以辅助开发人员快速完成编码；

第三步：创建格式对象 F3，引用查询对象 Q3 并嵌入表格，做相关的栏位、列格式和列表头设计。表头表尾可放置绑定参数的控件；

第四步：浏览 F3（或挂功能节点），设置参数，检查不同参数设置下数据与格式是否正确；

【FAQ】

1、 进行 SQL 整理对查询对象有什么要求？

答：该对象必须具有 SQL 设计，且强烈建议使用向导式 SQL 设计。

2、SQL 整理在何时何地进行？

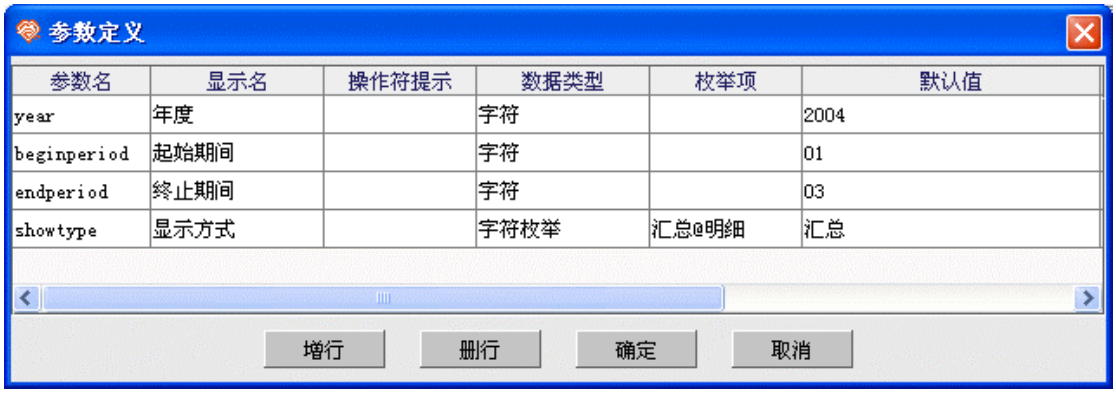
答：SQL 整理在服务器端进行，整理行为发生在数据库查询之前，多用于根据参数矫正 SQL。

【范例】

单位费用汇总或明细表——各单位在指定期间范围内的费用情况，查询汇总结果还是明细结果由参数控制。如果用户选择汇总，则报表中要求包括公司、期间、科目、汇总发生额等信息；如果用户选择明细，则报表中要求包括公司、期间、科目、明细发生额和分录摘要等信息。用户可以根据年度和期间范围进行查询。

1、 创建查询对象，编码为 glsumordetail，名称为费用汇总或明细；

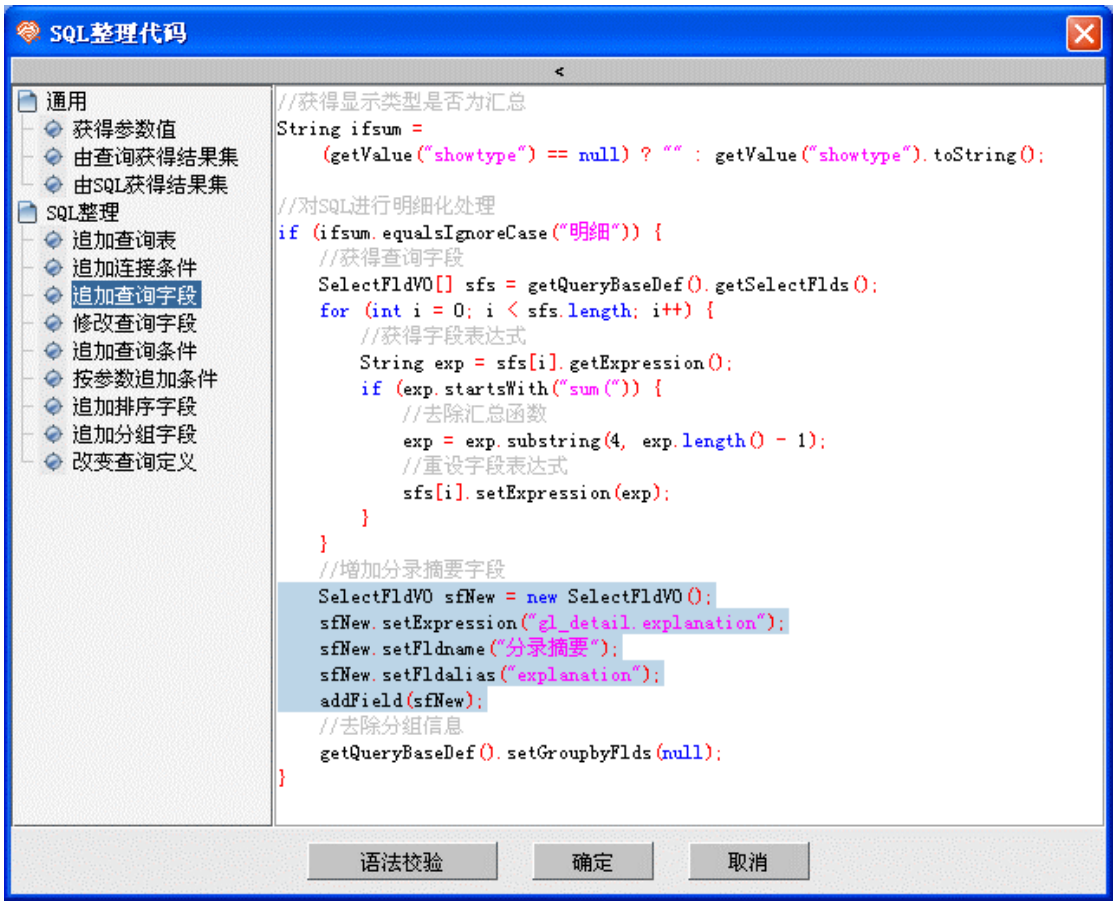
2、进行参数定义，增加三个字符型参数（年，起始期间，终止期间）和一个枚举型参数（显示类型，可选值为“汇总”和“明细”）；



3、进行 SQL 向导设计，内容与模式 2.2 的范例完全相同（我们在下面列出查询字段的界面，主要是方便读者把字段表达式与 SQL 整理中的代码处理结合起来理解，因为整理代码有很强的针对性）；

4、在筛选条件页签增加待定条件，内容与模式 2.2 的范例完全相同；

5、进行 SQL 整理设计——这段整理代码的思路是：首先获得用户设置的显示类型是“汇总”还是“明细”，如果是后者，则继续获得静态查询定义的查询字段数组（SelectFldVO[]），找出其中带有汇总函数的字段表达式，将汇总函数以及分组字段去掉，然后再增加一个新的查询字段，对应凭证分录表的摘要信息。

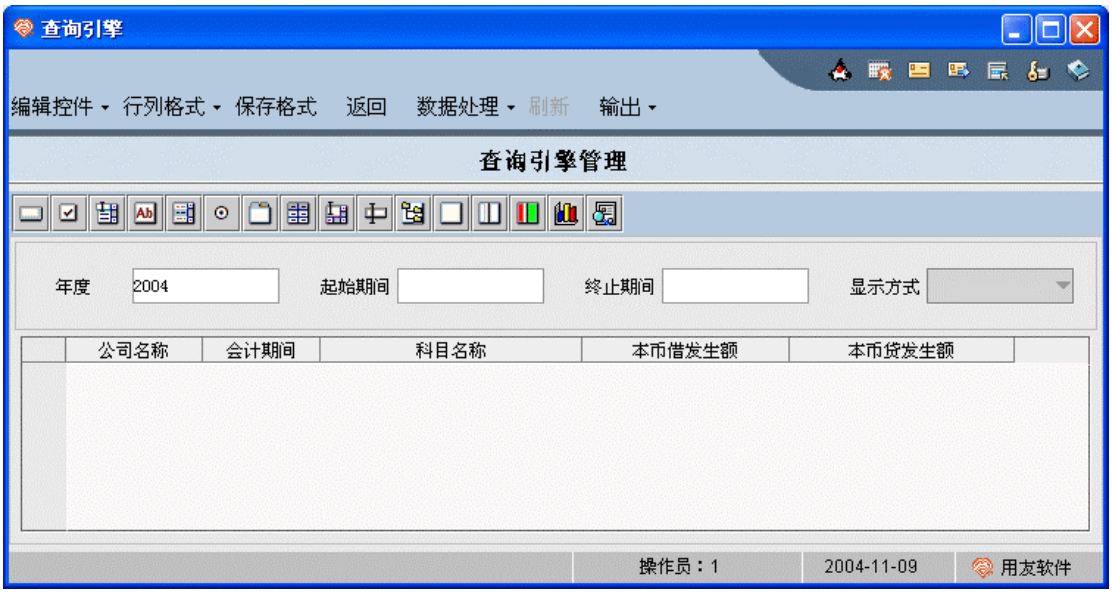


这段代码中一部分为左侧的代码向导树生成，其余部分则为直接手工编写。比如第一行代码就是由通用向导中的“获得参数值”生成：

而阴影部分（增加分录字段的代码片段）则由 SQL 整理向导中的“追加查询字段”生成：

6、创建格式对象，编码为 glsumordetail，名称为“费用汇总或明细表”。引用查询“费用汇总或明细”；

7、进行格式设计：添加表格和面板控件，表格停靠于中部，绑定数据集费用明细或汇总，面板停靠于北部，采用流式布局，在面板上放三个文本框控件，分别绑定三个字符型参数，再放置一个下拉框控件，绑定枚举型参数（显示类型）；



8、浏览报表（先设置参数“显示方式”的取值为“汇总”）；

	公司名称	会计期间	科目名称	本币借发生额	本币贷发生额
1	无锡分公司	01	长江工程公司	8030.0000	4850.0000
2	无锡分公司	01	长江集团总部	17523.4245	11708.0000
3	无锡分公司	01	长江政务公司	130.0000	8.0000
4	无锡分公司	01	贷款	74.0000	331.7500
5	无锡分公司	01	定期存款	0.0000	3.0000
6	无锡分公司	01	东莞电子制造厂	14568.6500	22696.8400
7	无锡分公司	01	工行存款	14655.5000	44943.8145
8	无锡分公司	01	广州结算中心	574.7500	139.0000
9	无锡分公司	01	金融机构借款	11446.5800	67.5000
10	无锡分公司	01	利息费用	2275.8333	0.0000
11	无锡分公司	01	内部贷款利息	0.0000	1000.0000
12	无锡分公司	01	上海结算中心	0.0000	2870.8333
13	无锡分公司	01	外部贷款	20000.0000	0.0000
14	无锡分公司	01	外部单位	720.0000	1380.0000
15	无锡分公司	02	长江集团总部	7000.0000	0.0000
16	无锡分公司	02	工行存款	0.0000	7000.0000
17	苏州分公司	01	本年利润	0.0000	181.6144
18	苏州分公司	01	财务费用	1000.0000	1000.0000
19	苏州分公司	01	待摊费用	1.2000	1000.0000
20	苏州分公司	01	固定资产	34530.0000	29.6100

9、再设置参数“显示方式”的取值为“明细”然后浏览报表，或者直接在浏览态切换表头的下拉框控件为“明细”选项；

3. 交叉类报表

数据库当中的表往往是象下面这样存储的：

姓名	月份	销售业绩
张三	1 月	2000
张三	3 月	5000
李四	2 月	3000
李四	3 月	4000

但用户要求的报表往往是象下面这样的：

姓名	1 月	2 月	3 月
张三	2000		5000
李四		3000	4000

类似这样，把原始数据中一列（或几列，通常有分组意义）数据的不同取值维持在表的左侧不动（比如上表中的姓名），把另外一列（或几列，通常有分类意义）数据的不同取值旋转到行的方向成为列标题（比如上表中的月份），再把另外一列（或几列，通常有统计意义）数据对应汇总到旋转出来的列标题下面（比如上表中的业绩），就形成了交叉表。

在上述过程中，我们根据交叉规则把原始数据中的列分成了三种，第一种称为交叉行（其数据交叉后出现在行上），第二种称为交叉列（其数据交叉后出现在列头上），第三种称为交叉值（交叉前行、列、值的数据出现在一行上，交叉后值的数据出现在行、列的交叉点上）。

交叉报表比普通报表提出了很多新的问题，比如：数据交叉如何实现，交叉后列数目的不确定性如何处理，列的顺序如何控制等等。根据实现方法的不同，以及能否预先确定交叉后的列结构，本章节提供了以下三种交叉报表模式的解决方案。

3.1 列向拼接查询型报表（复合查询）

【概述】

报表在列向可以被划分成几个分组，每组由若干个列构成，每个列分组的数据能够通过 SQL 查询获得，而不同列分组的数据出现在同一行的前提是它们对应于同一个行 ID。即报表数据可以通过多个查询结果集相互连接的方式获得。

【应用场景】

开发人员需要从某个具有分类意义的列中提取若干不同的取值，将它们旋转成为列标题，再对另外一些有统计意义的数值列进行汇总。用户表样明确规定了这些列标题及其顺序，因此交叉后的列结构是事先能够确定的，无论这些列是否包含有效的统计数据，它们都将在报表中存在，因此我们称之为动态行固定列报表。

【适用性和不适用性】

适用于：

- 能够预先确定交叉后列结构的报表；
- 报表的每行数据都具有一个唯一的行 ID，该 ID 可能是某个字段或某几个字段的组合；

不适用于：

- 动态列结构的交叉报表；

【解决方案】

利用复合查询解决固定列结构的交叉应用，每个交叉列对应一个查询，而用于连接多个查询的行 ID 正是交叉行字段的组合。

【开发步骤】

第一步：分析交叉报表的结构，找出交叉行字段、交叉列字段和交叉值字段。假定交叉列字段有 n 个（因为是固定列结构交叉，所以 n 是定数）；

第二步：创建 n 个查询对象分别对应于 n 个交叉列字段，通常这些对象会通过确定筛选条件与列标题挂钩；

第三步：创建复合查询对象 Q4，通过增加临时表的方式把上述 n 个查询对象选为查询表，这些临时表的连接条件就是交叉行字段的组合，而 Q4 的参数应该是 n 个查询对象的参数的并集，这样才能保证 Q4 的参数设置能够分发给各个被引用查询；

第四步：创建格式对象 F4，引用查询对象 Q4 并嵌入表格，做相关的栏位、列格式和列表头设计。表头表尾可放置绑定参数的控件；

第五步：浏览 F4（或挂功能节点），设置参数，检查数据与格式是否正确；

【FAQ】

1、 定义复合查询有什么注意事项？
答：首先要保证复合查询和各个子查询都使用相同的执行数据源（对于单数据源版查询引擎无此问题），其次如果修改了子查询的定义要保证复合查询定义中的一致性。

【范例】

部门员工民族统计表——要求统计指定公司下各部门中不同民族的员工数，民族预先指定为汉族、满族和蒙古族，报表中包含部门编码、部门名称、汉族人数、满族人数和蒙古族人数。请注意，这里作为交叉行并可以承担行 ID 角色的字段是部门信息，作为交叉列并可以承担子查询过滤角色的字段是民族，作为交叉值的字段是人数。

部门编码	部门名称	人数		
		汉族	满族	蒙古族

1、 创建查询对象，编码为 han，名称为汉族，在筛选条件页签设置确定条件“民族=’汉族’”，用于查询各部门的汉族员工人数；

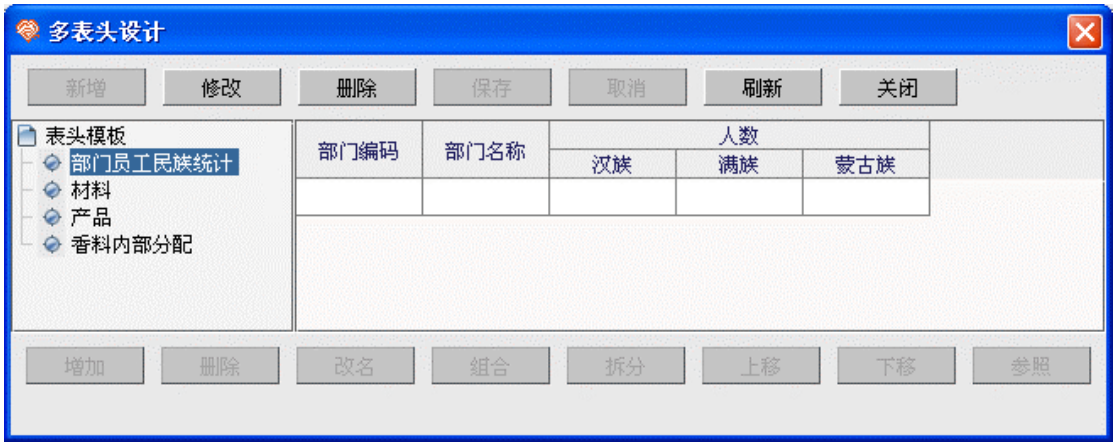
2、 复制查询对象 han，修改编码为 man，名称为满族，修改确定条件为“民族='满族'”，用于查询各部门的满族员工人数；

3、 复制查询对象 han，修改编码为 menggu，名称为蒙古族，修改确定条件为“民族='蒙古族'”，用于查询各部门的蒙古族员工人数；

4、 创建查询对象，编码为 nationalityA，名称为部门员工民族统计 A，选择表为部门档案和上述三个查询对象对应的临时表，利用部门主键字段连接。部门档案提供部门编码和名称信息，三个子查询分别提供三个民族的员工人数。我们之所以没有在子查询中直接查出部门编码和名称再在复合查询中利用部门编码进行连接，是因为这样做可能导致最终数据只列出拥有汉、满、蒙族员工的部门，而不列出全部的部门；

5、创建格式对象，编码为 nationalityA，名称为“部门员工民族统计表 A”；引用查询“部门员工民族统计 A”；

6、进行格式设计：添加表格控件，绑定数据集部门员工民族统计 A，进行列表头设计与合计设置；



7、浏览报表；

	部门编码	部门名称	人数		
			汉族	满族	蒙古族
1	01	直属职能部门	0	0	1
2	0101	总裁办	11	2	2
3	0103	人力资源部	7	0	1
4	0106	财务部	5	0	2
5	0108	行政部	6	0	1
6	03	ERP事业部	1	0	0
7	0301	ERP产品管理部	2	0	1
8	0302	ERP产品市场部	9	0	0
9	0304	ERP财务产品部	14	0	0
10	0305	ERP供应链产品部	11	0	2
11	0307	ERP生产制造产品部	10	0	0
12	0312	ERP服务支持部	7	0	1
13	05	HR事业部	1	0	0
14	0501	HR产品开发部	20	3	0
15	0502	HR产品市场部	4	0	0
16	0503	HR服务支持部	5	0	0
17	aa	商务部	0	2	0
18	bb	维修部	0	1	0
19	合计		113	8	11

3.2 列向分支统计型报表（CASE-WHEN）

【概述】

严格意义上来说，本模式与上一模式面向的是同一类报表，即预先能够确定交叉后列结构的交叉报表，或称动态行固定列交叉表。但本模式采用了另一种解决方案——结合 SQL 的分支函数和汇总函数，把交叉列生成的各个列分组写到一个查询当中，简化了复合查询方案需要写多个子查询对象的工作，但要求交叉值为可汇总的数值型字段。

【应用场景】

同上一模式。

【适用性和不适用性】

适用于：

- 能够预先确定交叉后列结构的报表；
- 报表的每行数据都具有一个唯一的行 ID，该 ID 可能是某个字段或某几个字段的组合；

不适用于：

- 动态列结构的交叉报表；
- 交叉值不能用 SQL 汇总函数统计的报表；

【解决方案】

CASE-WHEN 函数具有以下两种语法：

1、 简单 CASE 函数：

```
CASE  输入表达式
      WHEN  取值 1 THEN  返回表达式 1
      WHEN  取值 2 THEN  返回表达式 2
      ……
      ELSE  返回表达式 n
 END
```

2、 CASE 搜索函数：

```
CASE
      WHEN  条件判断表达式 1 THEN  返回表达式 1
      WHEN  条件判断表达式 2 THEN  返回表达式 2
      ……
      ELSE  返回表达式 n
 END
```

假定某张表 T 的数据如下所示：

corp	name	sex
A 公司	韩千穗	女
A 公司	智银圣	男
A 公司	金晓光	女
A 公司	王丽娜	女
B 公司	金贤成	男
B 公司	姜希灿	女

那么执行以下 SQL 语句

```
SELECT corp, (CASE sex WHEN '男' THEN 1 ELSE 0 END) AS male,
       (CASE sex WHEN '女' THEN 1 ELSE 0 END) AS female
FROM T
```

将获得如下结果集：

corp	male	female
A 公司	0	1
A 公司	1	0

A 公司	0	1
A 公司	0	1
B 公司	1	0
B 公司	0	1

如果我们关注的是交叉统计结果的话，只需将上述 SQL 改造为

```
SELECT corp, SUM(CASE sex WHEN '男' THEN 1 ELSE 0 END) AS male,
        SUM(CASE sex WHEN '女' THEN 1 ELSE 0 END) AS female
FROM TEM_CASE
GROUP BY corp
```

则将查出以下结果：

corp	male	female
A 公司	1	3
B 公司	1	1

也就是说 SUM(CASE WHEN)函数能够实现交叉统计的效果，而且一个 SQL 就能达到复合查询方案多个 SQL 的作用。在复合查询方案中出现于各子查询的确定筛选条件，就相当于 CASE-WHEN 方案中出现在 CASE 语句中的条件判断表达式。

【开发步骤】

第一步：分析交叉报表的结构，找出交叉行字段、交叉列字段和交叉值字段。假定交叉行字段有 m 个，交叉列字段有 n 个（因为是固定列结构交叉，所以 n 是定数）；

第二步：创建查询对象 Q5，定义 m 个字段表达式对应交叉行字段，再定义 n 个带有 SUM(CASE WHEN)的字段表达式对应交叉列字段，其中输入表达式为交叉列字段，取值为预先确定的交叉列字段取值，返回表达式为交叉值字段或其统计表达式，ELSE 分支返回 0；

第三步：创建格式对象 F5，引用查询对象 Q5 并嵌入表格，做相关的栏位、列格式和列表头设计。表头表尾可放置绑定参数的控件；

第四步：浏览 F5（或挂功能节点），设置参数，检查数据与格式是否正确；

【FAQ】

1、 CASE WHEN 语法的条件判断表达式能否使用非等值的表达式？
答：在 SQLSERVER 下没有问题，在 ORACLE 下应利用大于等于或小于等于比较符取代大于或小于比较符。

2、 是否有办法提高 SUM(CASE WHEN)的执行效率？
答：可考虑在 CASE-WHEN 中选择性高的字段上添加索引。

【范例】

部门员工民族统计表——要求与模式 3.1 完全相同。

1、 创建查询对象，编码为 nationalityB，名称为部门员工民族统计 B，请注意查询字段表达式中三处 CASE-WHEN 函数的写法：

sum(case bd_defdoc.docname when '汉族' then 1 else 0 end) （汉族人数）
sum(case bd_defdoc.docname when '满族' then 1 else 0 end) （满族人数）
sum(case bd_defdoc.docname when '蒙古族' then 1 else 0 end) （蒙古族人数）

2、创建格式对象，编码为 nationalityB，名称为“部门员工民族统计表 B”。引用查询“部门员工民族统计 B”；

3、完全类似于上一模式做格式设计：添加表格控件，绑定数据集部门员工民族统计 B，进

行列表头设计与合计设置；

4、浏览报表；

	部门编码	部门名称	人数		
			汉族	满族	蒙古族
1	01	直属职能部门	0	0	1
2	0101	总裁办	11	2	2
3	0103	人力资源部	7	0	1
4	0106	财务部	5	0	2
5	0108	行政部	6	0	1
6	03	ERP事业部	1	0	0
7	0301	ERP产品管理部	2	0	1
8	0302	ERP产品市场部	9	0	0
9	0304	ERP财务产品部	14	0	0
10	0305	ERP供应链产品部	11	0	2
11	0307	ERP生产制造产品部	10	0	0
12	0312	ERP服务支持部	7	0	1
13	05	HR事业部	1	0	0
14	0501	HR产品开发部	20	3	0
15	0502	HR产品市场部	4	0	0
16	0503	HR服务支持部	5	0	0
17	aa	商务部	0	2	0
18	bb	维修部	0	1	0
19	合计		113	8	11

3.3 动态行列交叉型报表（旋转交叉）

【概述】

与前两种交叉模式不同，本模式无法预先确定交叉后的列结构，而是根据数据完全真实地展现交叉结果，属于动态行动态列应用。这样就存在一个设计态列格式与运行态列结构之间的矛盾，因为设计态还没有查询数据，不可能预知运行态到底会生成哪些列，因此作为此类模式动态体现交叉结果的代价，开发人员无法对交叉值字段对应的列做精确的格式设计。

【应用场景】

在描述清楚交叉规则（指定交叉行、交叉列、交叉值字段）之后，报表要求全动态地实现数据交叉。多层列表头常用于辅助体现交叉的维度，且交叉规则能够在运行态重新设定。

【适用性和不适用性】

适用于：

- 无法预先确定交叉后列结构的报表；

不适用于：

- 对交叉值对应列有显示顺序等要求的报表；

- 对交叉值对应列有界面统计功能（如小计、图表等）预设置要求的报表；

【解决方案】

设计态使用查询模型 SQL 向导设计中的旋转交叉设置功能来描述初始交叉规则，运行态使用表格表头右键菜单中的交叉功能实现再次交叉。交叉设置界面中的交叉砝码的作用是，描述交叉值字段对应的列出现在交叉列展开后的哪个层次上。

以我们前面做过的费用汇总表为例，采用以下的交叉设置：

将得到以下的交叉结果：

公司名称	本币借发生额			本币贷发生额		
	01	02	03	01	02	03
常州分公司	191943.076	37243.10789999999	8.5292	216305.4239	41105.921599999994	8.5292
无锡分公司	89998.7378	7000.0		89998.7378	7000.0	
苏州分公司	156098.2255	60044.0		199038.4311	95040.5	

如果更改交叉设置中交叉砝码的位置：

则将获得以下交叉结果：

公司名称	01		02		03	
	本币借发生额	本币贷发生额	本币借发生额	本币贷发生额	本币借发生额	本币贷发生额
常州分公司	191943.076	216305.4239	37243.10789999999	41105.921599999994	8.5292	8.5292
无锡分公司	89998.7378	89998.7378	7000.0	7000.0		
苏州分公司	156098.2255	199038.4311	60044.0	95040.5		

【开发步骤】

第一步：分析交叉报表的结构，找出交叉行字段、交叉列字段和交叉值字段；

第二步：创建查询对象 Q6，正常定义 SQL（以获得交叉前结果集为准），在交叉页签的旋转交叉设置中指定交叉行、交叉列、交叉值以及交叉砝码的位置；

第三步：创建格式对象 F6，引用查询对象 Q6 并嵌入表格；

第四步：浏览 F6（或挂功能节点），设置参数，检查数据与格式是否正确；

【FAQ】

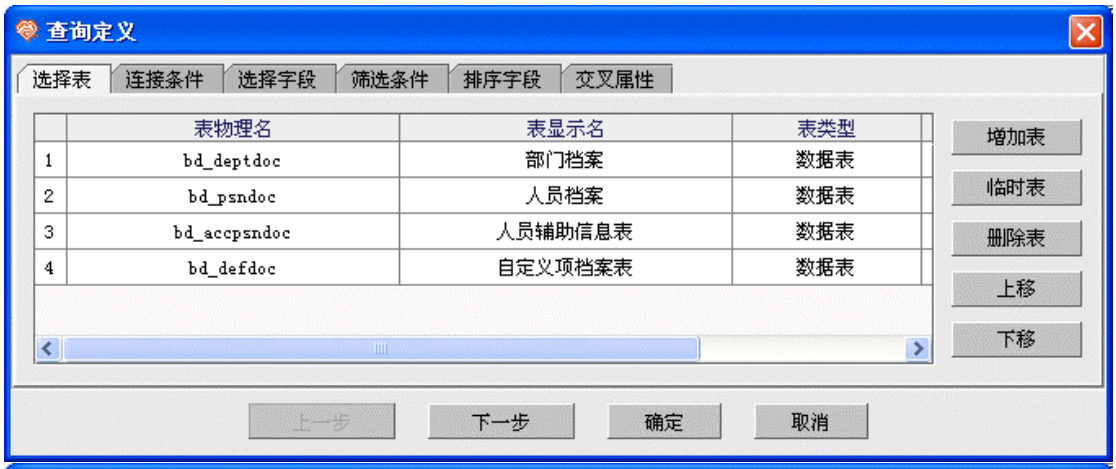
1、对绑定旋转交叉数据集的表格绑定列表头是否有效？
答：无效，系统将根据交叉设置自动生成列表头。

【范例】

部门员工民族统计表——要求与模式 3.1 完全相同。

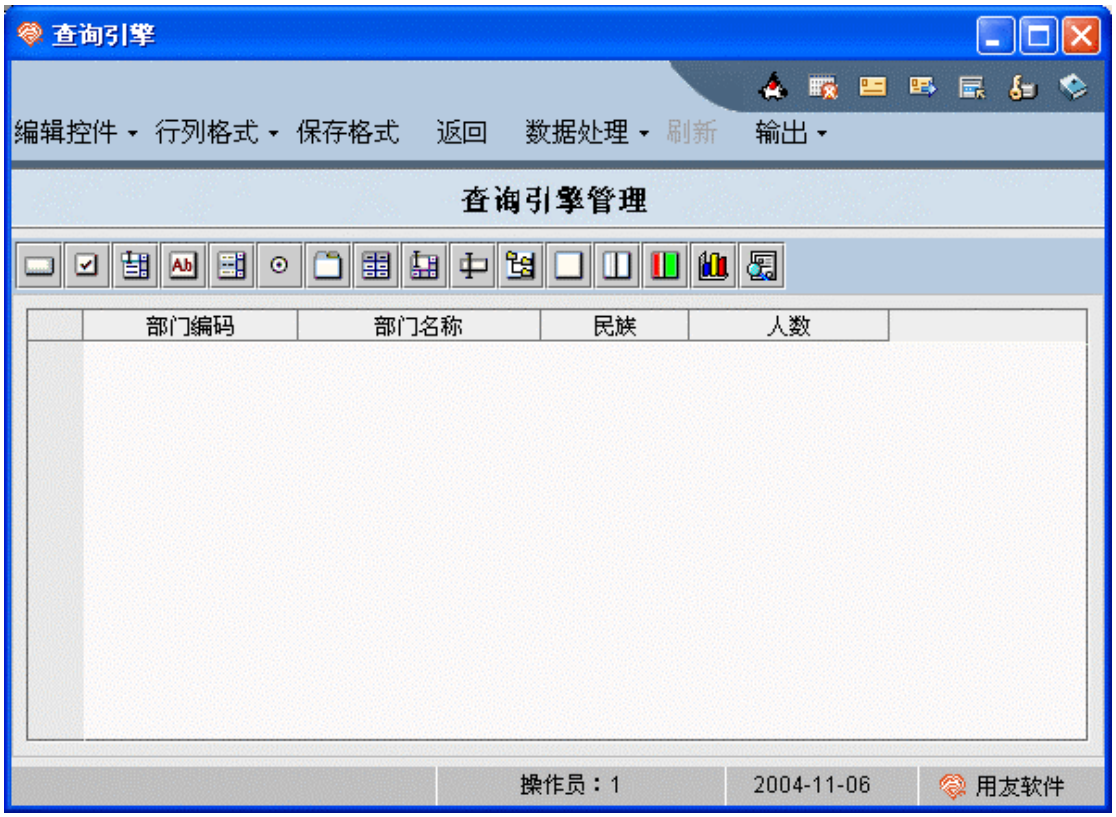
部门编码	部门名称	人数			
		民族 1	民族 2	……	民族 n

1、 创建查询对象，编码为 nationalityC，名称为部门员工民族统计 C，查询字段中按部门、民族对人员计数，旋转交叉设置中指定部门编码、名称为交叉行，民族为交叉列，人数为交叉值：



2、创建格式对象，编码为 nationalityC，名称为“部门员工民族统计表 C”。引用查询“部门员工民族统计 C”；

3、进行格式设计：添加表格控件，绑定数据集部门员工民族统计 C；



4、浏览报表，请注意所有民族员工的人数均列示在表格中；

	部门编码	部门名称	人数			
			汉族	满族	苗族	蒙古族
1	01	直属职能部门				1
2	0101	总裁办	11	2	1	2
3	0103	人力资源部	7			1
4	0106	财务部	5			2
5	0108	行政部	6			1
6	03	ERP事业部	1			
7	0301	ERP产品管理部	2			1
8	0302	ERP产品市场部	9			
9	0304	ERP财务产品部	14			
10	0305	ERP供应链产品部	11			2
11	0307	ERP生产制造产品部	10			
12	0312	ERP服务支持部	7			1
13	05	HR事业部	1			
14	0501	HR产品开发部	20	3		
15	0502	HR产品市场部	4			
16	0503	HR服务支持部	5			
17	aa	商务部		2		
18	bb	维修部		1		

4. 投影类报表

本手册中提到的投影类报表，就是固定行固定列的报表。所谓投影是指其每个单元格内部存放的内容与该单元的坐标（行列位置）存在直接的关系。

4.1 单元格依赖于行列条件的报表（投影交叉）

【概述】

	列条件 1	……	列条件 j	……	列条件 n
行条件 1	(1, 1)		(1, j)		(1, n)
……					
行条件 i	(i, 1)		(i, j)		(i, n)
……					
行条件 m	(m, 1)		(m, j)		(m, n)

本类报表具有以下特点：

1、 具有固定的行列结构：m 行，n 列。交叉结果严格遵循表样格式，其行列结构不会受查询条件和查询数据的影响；

2、 每行、每列均对应一个筛选条件，而表体第 i 行第 j 列的内容可以根据 SQL 语句“select 统计函数(字段) from 表 where (行条件 i) and (列条件 j)”查出，且是个唯一的值（这里的表可能由多张表或视图连接在一起得到）；

在人事统计报表当中有很多此类应用，比如列头包括各种学历结构，行头包括各种职务序列，那么出现在本科学历所在列和处长所在行的交叉点上的内容，可能就是所有本科学历处长的人数，用 SQL 语言表示就是：

select count(人员) from 人事视图 where 学历='本科' and 职务='处长'

【应用场景】

报表表样形态固定，行列顺序均不能随意更改，数据来源比较一致，表体数据由所在单元向行、列投影得到的限制条件叠加确定。在格式设计方面，不仅对列表头，而且对行表头都有一定的要求。

【适用性和不适用性】

适用于：

- 表单元数据由固定 SQL 结合行列对应的 WHERE 条件唯一决定；
- 多行头报表；

不适用于：

- 不确定行结构或不确定列结构的报表；
- 各列分组来自不同业务系统且不能共用一个 SQL 的报表；

【解决方案】

利用查询引擎提供的投影交叉方案，在 SQL 向导设计中定义一个不完整的查询 SQL，以及两组 WHERE 条件，一组对应行头，一组对应列头。

前面说过，本类报表的表单元数据由以下 SQL 确定：

SQL_{ij} = select 统计函数(字段) from 表 where (行条件i) and (列条件j)

我们把 where 前面的部分成为 SQL 主体，不难看出，一张 m 行 n 列的投影交叉表，就包含了 m*n 个 SQL 语句，但它们全都共用一个 SQL 主体。在查询对象的 SQL 向导设计当中，前五个页签用于描述 SQL 主体，因为每个单元格只能放一个值，所以主体部分通常只有一个查询字段表达式，虽然这部分可能查了很多张表，但这些表大多不服务于那个孤独的查询字段，而是服务于行列条件，最后一个页签（交叉属性）的投影交叉设置可以添加任意多的行列筛选条件，每个条件用不带 where 的 SQL 片段描述，比如“性别='女'”或者“1=1”。

由上可知，SQL 主体部分和行列条件部分单独存在都是不完整的，只有它们联合起来才能描述一组（m*n 个）完整的查询。

【开发步骤】

第一步：分析交叉报表的结构，找出行条件、列条件和对应于单元数据的统计表达式；

第二步：创建查询对象 Q7，在查询表、连接条件、查询字段、筛选条件和排序字段页签中定义 SQL 主体，在交叉属性页签定义行条件和列条件；

第三步：创建格式对象 F7，引用查询对象 Q7 并嵌入表格，进行行列表头设计；

第四步：浏览 F7（或挂功能节点），设置参数，检查数据与格式是否正确；

【FAQ】

1、投影交叉的查询定义是否只能定义一个查询字段表达式？
答：绝大部分应用都是这样的。如果定义了 n 个查询字段，每个列条件下对应的列都将翻 n 倍，这不是非常推荐的做法。

【范例】

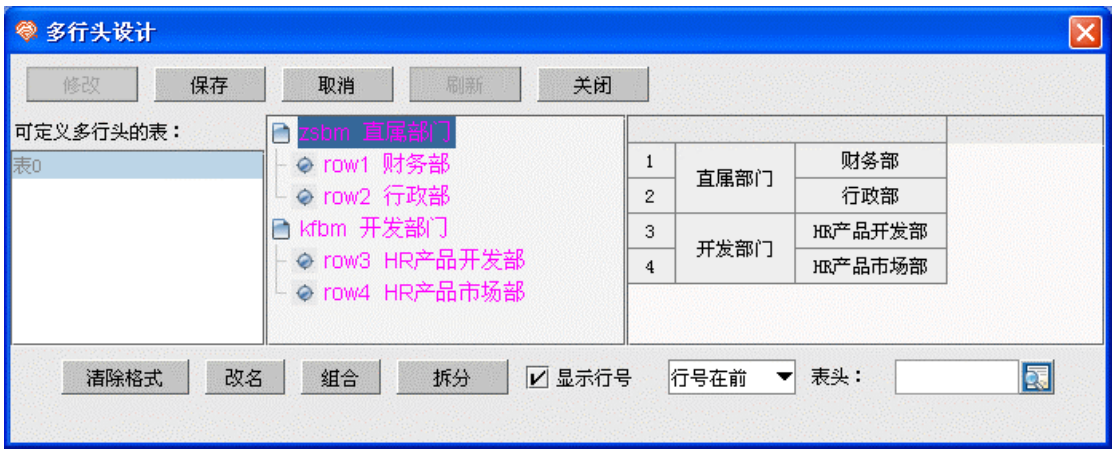
固定行固定列的部门民族员工统计表——要求显示指定部门指定民族的人数统计，显示顺序不能随便设置，格式支持存在多行头和行头小计。表样如下所示：

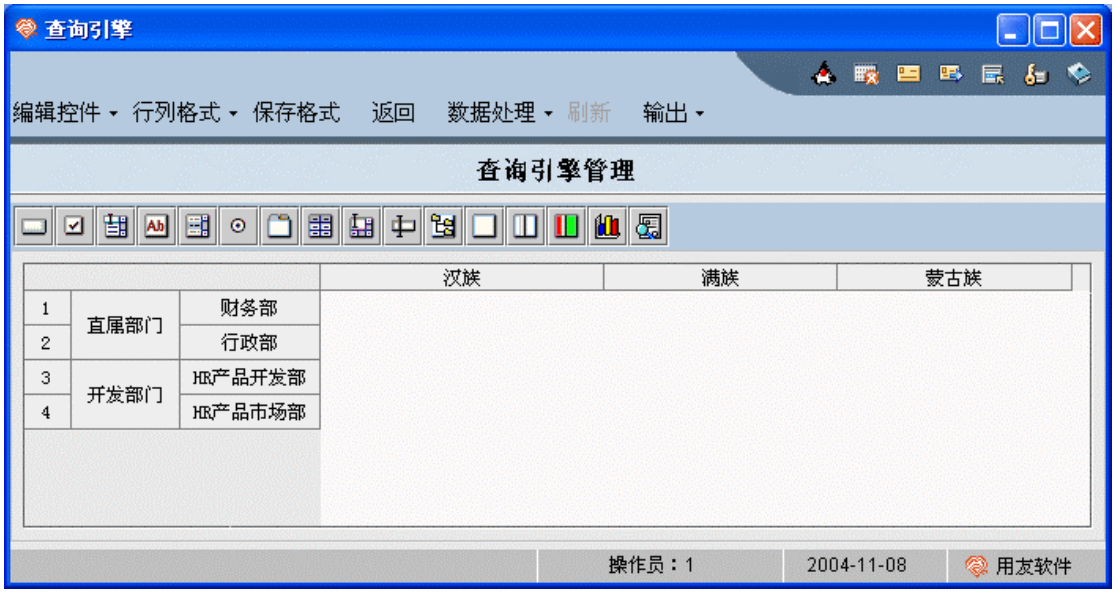
		汉族	满族	蒙古族
直属部门	财务部			
	行政部			
	小计			
开发部门	HR 产品开发部			
	HR 产品市场部			
	小计			
合计				

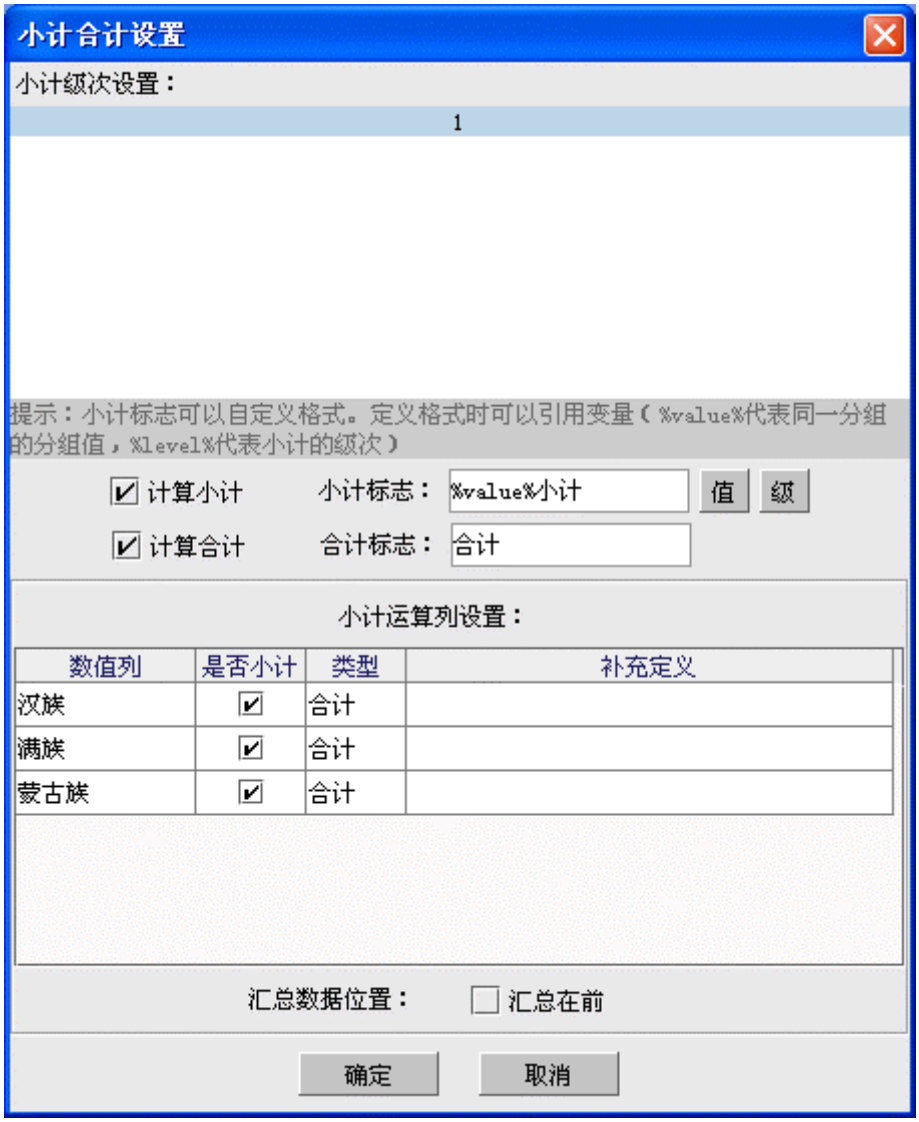
1、创建查询对象，编码为 nationalityD，名称为部门员工民族统计 D。仿照模式 3.3（旋转交叉）进行 SQL 设计，查询字段只保留人员计数表达式一项，交叉属性进行投影交叉设置，根据表样指定行列 WHERE 条件；

2、创建格式对象，编码为 nationalityD，名称为“部门员工民族统计表 D”。引用“查询部门员工民族统计 D”；

3、进行格式设计：添加表格控件，绑定数据集部门员工民族统计 D，做多行头设计和小计合计设置；




查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
查询引擎管理
汉族
满族
蒙古族
1
直属部门
财务部
2
行政部
3
开发部门
HR产品开发部
4
HR产品市场部
操作员：1
2004-11-08
用友软件


小计合计设置
小计级次设置：
1
提示：小计标志可以自定义格式。定义格式时可以引用变量（%value%代表同一分组的分组值，%level%代表小计的级次）
计算小计
小计标志：
%value%小计
值
级
计算合计
合计标志：
合计
小计运算列设置：
数值列
是否小计
类型
补充定义
汉族
合计
满族
合计
蒙古族
合计
汇总数据位置：
汇总在前
确定
取消

4、浏览报表；

5、运行态利用交叉设置实现行列旋转；

4.2 半录入半嵌入型报表（合并查询）

【概述】

经常能看到这样一类固定行列结构表样，在表的不同区域有不同的数据块（这些块之间甚至可能没有什么联系），另外一些区域还放着一些人为性很强的提示性文字（可能根本无法从数据库中查出来），象这样的无规律报表其实是报表开发中最难处理的。本模式采用的方案是对解决这类报表所做的一个初步尝试，它支持开发者把利用其它方式定义的查询嵌入到表体的某个区域，也可以把那些无法查出的文字直接录入到需要的位置。

【应用场景】

用户报表由若干矩形区域构成，每个区域或者可以用查询结果填充，或者可以手工填充。

【适用性和不适用性】

适用于：

- 独立查询数据与独立录入数据相结合的报表；
- 由共用一组参数的多个查询嵌入构成报表；

不适用于：

- 要求嵌入查询动态扩展的报表；
- 某嵌入查询的定义依赖于另一个嵌入查询查出的数据；
- 单元格之间存在公式依赖关系；
- 同一列数据存在不同数据类型；

【解决方案】

利用查询引擎提供的合并查询方案，可以自由增行（提供行名称）和增列（提供列名称和列数据类型），可以在任意单元格录入文字，可以在任意位置嵌入其它查询（提供此查询的占位情况，即左上角和右下角坐标），嵌入查询的定义与本方案无关。

【开发步骤】

第一步：分析报表结构，划分出若干可以通过查询结果填充的数据；

第二步：创建查询对象 Q8，进行合并查询设计，通过增行增列扩充可用区域，通过录入文字和嵌入查询填充内容。嵌入查询以当前选中单元作为左上角缺省坐标，如果嵌入投影交叉查询，则系统自动带出右下角缺省坐标，如果嵌入其它查询（动态行），则系统只自动带出右下角的列坐标；

第三步：创建格式对象 F8，引用查询对象 Q8 并嵌入表格，进行行列表头设计；

第四步：浏览 F8（或挂功能节点），设置参数，检查数据与格式是否正确；

【FAQ】

1、 系统如何处理嵌入查询指定的行列数与实际查出数据的行列数不符的情况？
答：多查出的部分被截断，多指定的区域自动置空。

2、 能否不显示行头？
答：暂时不能。

【范例】

仿手工填报的部门员工民族统计表，表样如下所示：

部门	汉族	满族	蒙古族
1、直属部门情况			
财务部	<人数>	<人数>	<人数>
行政部	<人数>	<人数>	<人数>
2、开发部门情况			
HR 产品开发部	<人数>	<人数>	<人数>
HR 产品市场部	<人数>	<人数>	<人数>

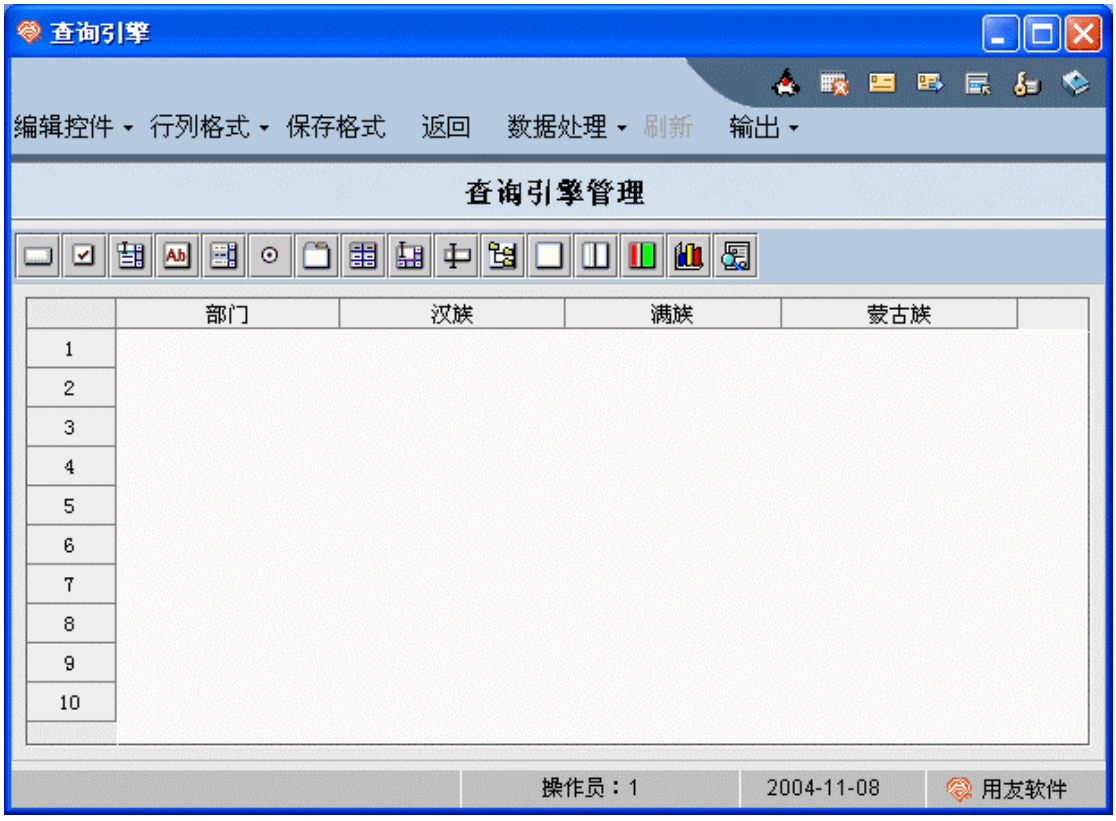
1、复制查询对象“部门员工民族统计 D”，修改编码为 directdept，名称为直属部门。交叉设置中删除与 HR 开发部门相关的两行；

2、复制查询对象“部门员工民族统计 D”，修改编码为 developdept，名称为开发部门。交叉设置中删除财务部和行政部两行；

3、创建查询对象，编码为 nationalityE，名称为部门员工民族统计 E。进行合并查询设计，插入 10 行 4 列，指定列名和列数据类型（第一列为字符型，后三列为整型），第一列录入部门信息，选中(列 1，行 2)单元嵌入查询“直属部门”，选中(列 1，行 6)单元嵌入查询“开发部门”；

4、创建格式对象，编码为 nationalityE，名称为“部门员工民族统计表 E”。引用查询“部门员工民族统计表 E”；

5、进行格式设计：添加表格控件，绑定数据集部门员工民族统计 E；


查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
查询引擎管理
部门
汉族
满族
蒙古族
操作员：1
2004-11-08
用友软件

6、报表浏览；


查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
部门员工民族统计表E
部门
汉族
满族
蒙古族
1、直属部门情况
财务部 5 0 2
行政部 6 0 1
2、开发部门情况
HR产品开发部 20 3 0
HR产品市场部 4 0 0
操作员：1
2004-11-08
用友软件

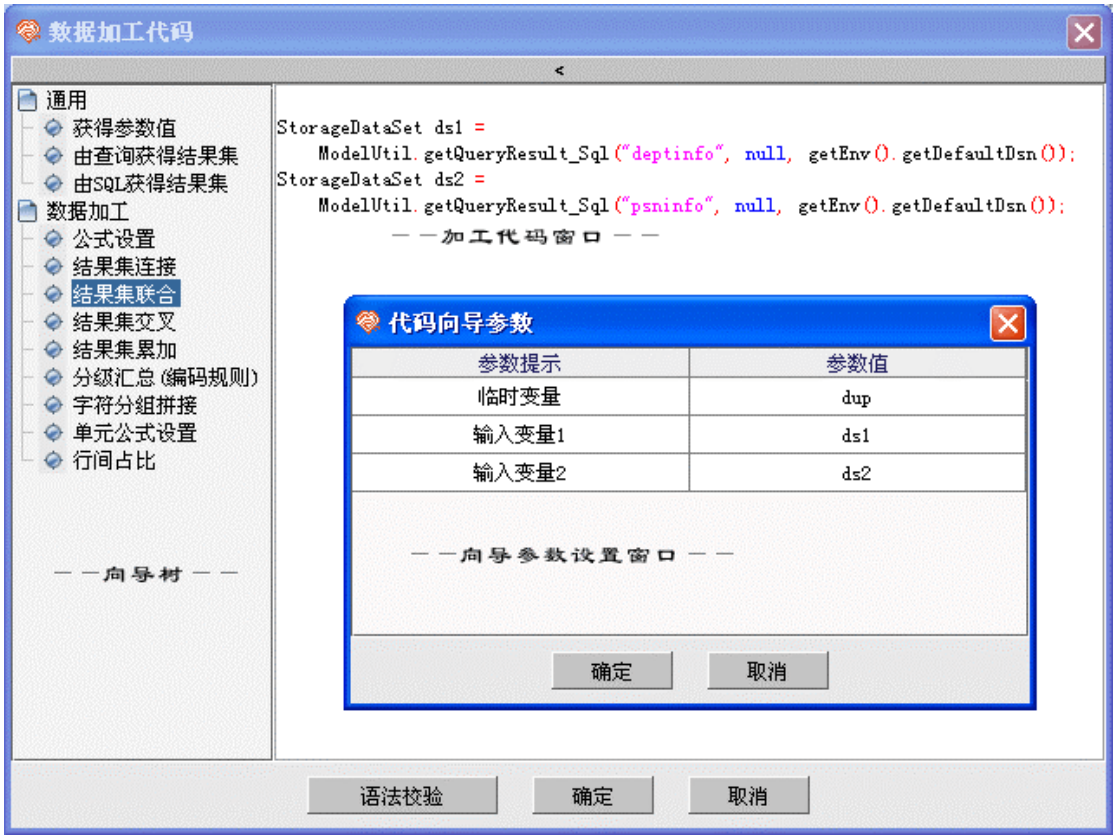
5. 算法类报表

查询引擎作为自定义查询体系的开发工具，定位于尽量让使用者少写代码或不写代码，但我们必须意识到，大量用户报表具有极高的复杂度，不是依靠简单查询或者复合查询就能解决问题的，而需要经过大量的业务代码的运算才能获得最终结果。因此查询引擎提供了开发人员对查询结果进行数据加工的方案，以弥补有限的 SQL 功能。在加工界面上，用户既可以利用查询引擎内置的多种常用算法，也可以调用自己编写的服务器端代码，或者把自己设计的加工算法注册到查询引擎共享给其他开发人员使用，实现代码的可复用性。加工算法统一采用 JAVA 语法的代码片段，需要通过数据加工才能开发出来的报表通称为算法类报表。

5.1 数据加工预备知识

数据加工是指某个（或某几个）结果集在经过一系列加工算法的变换后得到另一个结果集，而后者通常是业务上要求的展现结果。相关的算法又称为加工滤镜，滤镜通常支持输入多个结果集和输出一个结果集，并可以嵌套使用。在第一章提到过，查询引擎使用的结果集是第三方数据结构 StorageDataSet。

数据加工的界面分为两部分，左侧为代码向导树，右侧为代码编写窗口。代码向导用于辅助编码，双击向导树节点将弹出算法参数设置框，根据参数提示设定参数值后将生成加工代码片段，专业开发人员可以对生成的代码做进一步的修改，但向导生成代码的动作是单向的。



加工算法和算法向导均可获得扩展。编写加工算法可参考 nc.vo.dbbase.tools 提供的滤镜式实

现方案，如果想注册加工向导，可以在 pub_codewizard 中手工初始化数据，各字段含义由名称易知，向导类均继承自抽象类 nc.vo.pub.codingwiz.IcodeWiz，具体向导的实现代码可参见 nc.vo.pub.codingwiz 下的类。

加工代码中可以调用以下两个重要接口方法：

void setDataSet(StorageDataSet);

StorageDataSet getDataSet();

第一个方法用于设置某个数据集为当前数据集，如果该方法出现在加工代码最后，那么作为参数的数据集就将成为报表展现的最终数据结果；第二个方法用于当前加工状态下的数据集（数据集可能经过多个加工步骤才能达到目标结果，而 getDataSet()可用于传递中间结果）。

虽然查询引擎能够展现任意动态结构的数据集，但仍然希望开发人员在设计带有数据加工的查询对象时尽量遵循“列结构稳定”原则，这样做可以保证在格式设计态进行更多有效工作。假定我们的静态查询结果有 A、B 两列，然后准备通过数据加工的公式列算法增加一列数值列 C（它可能是 A 和 B 的函数），那么最佳做法是在静态查询设计时就预留出 C 的位置，即查询字段包括 A、B、C 三列，其中 C 列的查询表达式设为 0 或 0.0（取决于该列的数据类型），然后在数据加工中设置 C 列的公式。这种预留字段然后利用算法设置该列取值（而不是动态增加该列）的做法，就遵循了“列结构稳定”原则，它使得我们能够在设计态设置 C 列的格式信息。

数据加工是高开发技能人员在查询引擎中充分发挥的战场，本章介绍的占比模式和程序送数模式只是数据加工的某些具体应用。以下对查询引擎内置加工算法做一简单介绍：

获得参数值

获得用户浏览本查询时对某个参数设置的值。

算法参数：

参数名：待获得参数的编码。

生成代码示例：

```
String par = (getValue("param1") == null) ? "" : getValue("param1").toString();
```

由查询获得结果集

获得某个查询对象的执行结果，用于进一步运算。

算法参数：

查询 ID：待获得结果集的查询 ID。

参数变量：待获得结果集的参数设置 Hashtable，键为参数名，值为参数 VO。

生成代码示例：

```
StorageDataSet ds = ModelUtil.getQueryResult_Sql("Q1", getHashParam(), “nc0717”);
```

公式设置

设置公式运算列。

算法参数：

公式列名：若此列名不存在则新增一列，否则将该列的内容更新为公式返回值。

返回值数据类型：字符，整数，小数。

公式：符合 NC 公式解析语法的公式字符串。

生成代码示例：

```
String[] strColNames = new String[] { "col1", "col2" };
int[] iDataTypes = new int[] { Variant.STRING, Variant.DOUBLE };
String[] strColFormulas = new String[] {
    "col1->getColValue( pub_datadict,display ,id ,'00000000001' )",
    "col2->col3+col4"
};
DataFormulateProvider dfp = new DataFormulateProvider(strColNames, iDataTypes, strColFormulas);
dfp.setDataProviders(new Object[] { ds });
setProvider(dfp);
```

结果集连接

将指定的两个结果集通过模拟数据库连接获得新的结果集。

算法参数：

连接模式：三种连接方式－内连接、左连接、右连接。

连接字段（逗号分隔）：连接字段。

保留字段（逗号分隔）：连接后结果集的保留字段。

生成代码示例：

```
DataSetJoinProvider djp = new DataSetJoinProvider(new String[] { "pk_corp" }, DataSetJoinProvider.INNER_JOIN);
djp.setDataProviders(new Object[] { ds1, ds2 });
djp.setReservedFields(new String[] { "pk_corp", "col1", "col2" });
setProvider(djp);
```

结果集联合

指定两个列结构相兼容的结果集，得到联合（union）后的结果。

算法参数：

生成代码示例：

```
DataUnionProvider dup = new DataUnionProvider();
dup.setDataProviders(new Object[] { ds1, ds2 });
setProvider(dup);
```

结果集交叉

将一个结果集按照交叉行、交叉列、交叉值生成旋转交叉后的结果集。

算法参数：

交叉行，交叉列，交叉值：参见模式 3 的说明，更详细内容可参阅交叉表说明文档。

生成代码示例：

```
String[] strCrsRows = new String[] { "send_date" };
String[] strCrsCols = new String[] { "send_id" ,"&type"};
String[] strCrsVals = new String[] { "number", "sum" };
DataCrossProvider dcp = new DataCrossProvider(strCrsRows, strCrsCols, strCrsVals);
dcp.setDataProviders(new Object[] { ds });
setProvider(dcp);
DataUnionProvider dup = new DataUnionProvider();
dup.setDataProviders(new Object[] { ds1, ds2 });
setProvider(dup);
```

结果集累加

支持两个结果集按照键累加值得到的结果，假设结果集 1 为：

K1	K2	V
1	1	10
1	3	20
2	1	30

结果集 2 为：

K1	K2	V
1	1	4
2	1	5

连接键为 K1、K2，则合并累加（相当于内连接后求和）后的结果为：

K1	K2	V
1	1	14
2	1	35

连接累加（相当于外连接后求和）后的结果为

K1	K2	V
1	1	14
1	3	20
2	1	35

算法参数：

连接模式：全连接或左连接。

连接字段（逗号分隔）：上述说明中的键。

累加字段（逗号分隔）：上述说明中的值。

生成代码示例：

```
DataSetSumProvider dsp = new DataSetSumProvider(new String[] { "pk_corp" }, new String[]
```

```
{ "col1", "col2" }, nc.vo.pub.rs.IResultSetConst.JOIN_MODE);
dsp.setDataProviders(new Object[] { ds1, ds2 });
setProvider(dsp);
```

分级汇总（编码规则）

假定某结果集由末级数据构成，本算法可根据其编码列按指定编码规则进行逐级汇总。

算法参数：

列名（逗号分隔）：进行汇总的数值列。

级次列：编码列。

编码规则（斜杠分隔）：例如：2/2/2/2。

汇总上限级次：对“2/2/2/2”的编码规则，上限级次为 0，整个结果集添加合计行。

汇总下限级次：对“2/2/2/2”的编码规则，下限级次为 4，每个末级添加小计行。

生成代码示例：

```
nc.vo.pub.report.CodeLevelResultSetContext clrsc = new nc.vo.pub.report.CodeLevelResultSetContext();
clrsc.setCols(new String[][] { { "number", "number" }, { "sum", "sum" } });
clrsc.setLevelCol("send_id");
clrsc.setLevelMode("2/2/2/2");
clrsc.setTopLevel(0);
clrsc.setBottomLevel(4);

DataLevelProvider dlp = new DataLevelProvider(clrsc);
dlp.setDataProviders(new Object[] { ds });
setProvider(dlp);
```

单元公式设置

对于固定行固定列的报表或可预测的动态报表，可以设置单元格的公式（包括依赖公式）。

算法参数：

查询 ID：提供结果集的查询对象 ID。

单元格公式：符合 NC 公式解析语法的公式。

公式返回数据类型：整数、小数。

生成代码示例：

```
StorageDataSet dsOld = ModelUtil.getQueryResult_Sql("Q2", null, "nc0717");
String[] strCellFormulas = new String[] {
    "[0,1]->[0,0]/[2,0]",
    "[1,1]->[1,0]/[2,0]",
    "[2,1]->[2,0]/[2,0]",
    "[3,1]->[3,0]/[5,0]",
};
int[] iFormulaTypes = new int[] {8, 8, 8, 8};
```

```
StorageDataSet dsNew = DatasetUtil.convDatasetByCell(dsOld, strCellFormulas, iFormulaTypes);
setDataSet(dsNew);
```

行间占比

计算动态行报表的行间占比。

算法参数：

分组排序列（可空，逗号分隔）：先按分组排序列进行分组排序，然后计算单元格与其所在分组小计值的比，如设空，则为单元格与其所在列合计值的比。

被除列（逗号分隔）：需要计算行间占比的列。

占比列（逗号分隔）：存放占比结果的列。

生成代码示例：

```
String[] strGroupCols = new String[] { "pk_dept" };
String[] strDividedCols = new String[] { "mennum", "womennum" };
String[] strPercentCols = new String[] { "menper", "womenper" };
RateComputeProvider rcp = new RateComputeProvider(strGroupCols, strDividedCols, strPercentCols, false);
rcp.setDataProviders(new Object[] { ds });
setProvider(rcp);
```

5.2 非投影类占比报表（普通占比）

【概述】

部门	男员工数	男员工占比	女员工数	女员工占比
部门 1	8	**？%**	2	**？%**
……	……	……	……	……
部门 n	……	……	……	……
合计	80		50	

上表就是一张占比报表，但相信细心的读者会提出疑问：第一行的占比到底是谁比谁？是部门 1 的男员工占所有部门男员工总数的比例（10%），还是部门 1 的男员工占部门 1 员工总数的比例（80%）？为了避免这种二义性，我们提出行间占比和列间占比的概念，上述第一种理解成为行间占比，即比例=本行数值/多行总和，第二种理解成为列间占比，即比例=本列数值/多列总和。本模式讨论的是动态行固定列报表的行间和列间占比问题。

【应用场景】

用户表样要求，统计单元取值占所属行分组小计结果的比例，或占所属列分组小计结果的比例，并显示在专门的占比列上。

【适用性和不适用性】

适用于：

- 动态行报表占比；

不适用于：

- 要求计算某单元与另一个错行错列单元的数值比例；

【解决方案】

行间占比采用数据加工内置的行间占比算法，列间占比采用内置的公式列算法。

【开发步骤】

第一步：创建查询对象，利用向导设计定义 SQL，并预留空列待占比结果填充；

第二步：在数据加工中获得上述查询结果集，利用行间占比算法处理行间占比问题，利用公式列算法处理列间占比问题；

第三步：创建格式对象，引用查询，设计格式。占比列可绑定百分比列格式；

第四步：浏览报表。

【范例】

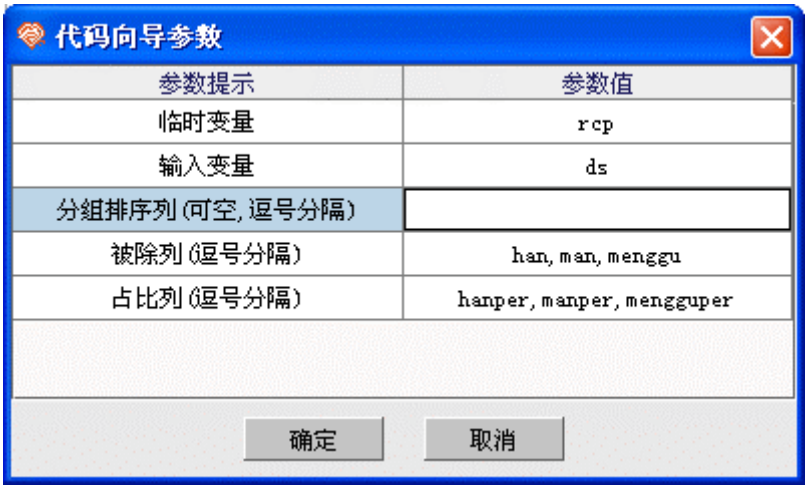
部门员工民族占比表（动态行），表样如下，要求利用两张表格分别统计行间和列间占比。

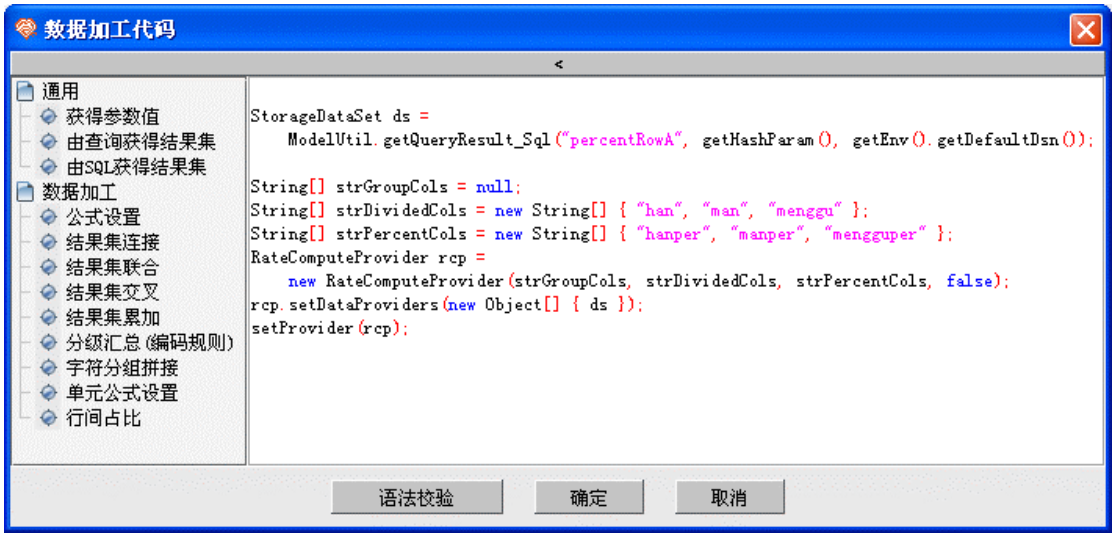
部门	汉族员工人数	汉族员工占比	满族员工人数	满族员工占比	蒙族员工人数	蒙族员工占比
部门 1						
……	……	……	……	……	……	……
部门 n						
合计						

1、 复制查询对象“部门员工民族统计 B”，修改编码为 percentRowA，名称为“部门员工民族行间占比 A”。修改向导设计的查询字段页签，增加三个预留字段（汉族占比、满族占比、蒙古族占比，字段表达式均为 0.0），准备用于填充加工出来的占比值；

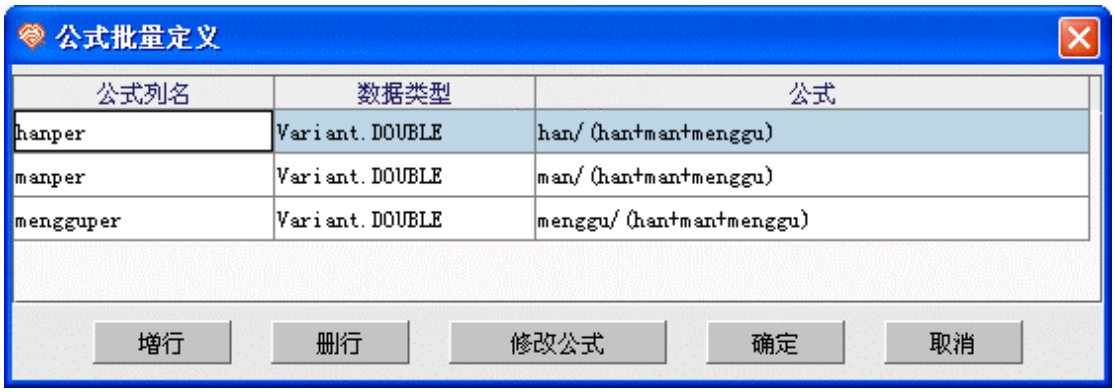
2、 复制上一步骤定义的查询对象“部门员工民族行间占比 A”，修改编码为 percentColA，名称为“部门员工民族列间占比 A”。这是因为这两个查询的加工前状态是一致的；

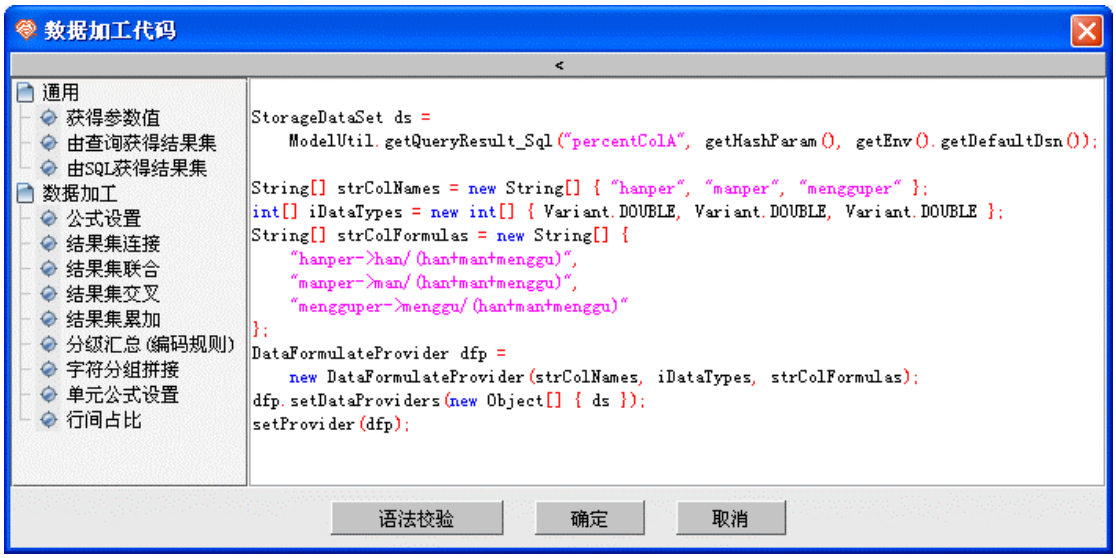
3、 对“部门员工民族行间占比 A”进行数据加工，首先利用“获得结果集”向导生成从 percentRowA 取数的代码，再利用“行间占比”向导生成计算行间占比的代码，注意向导参数设置中均使用字段别名，另外，此处采用的滤镜方式无须通过 getDataSet()和 setDataSet()传递加工数据集，因为它们已经被滤镜内部处理；



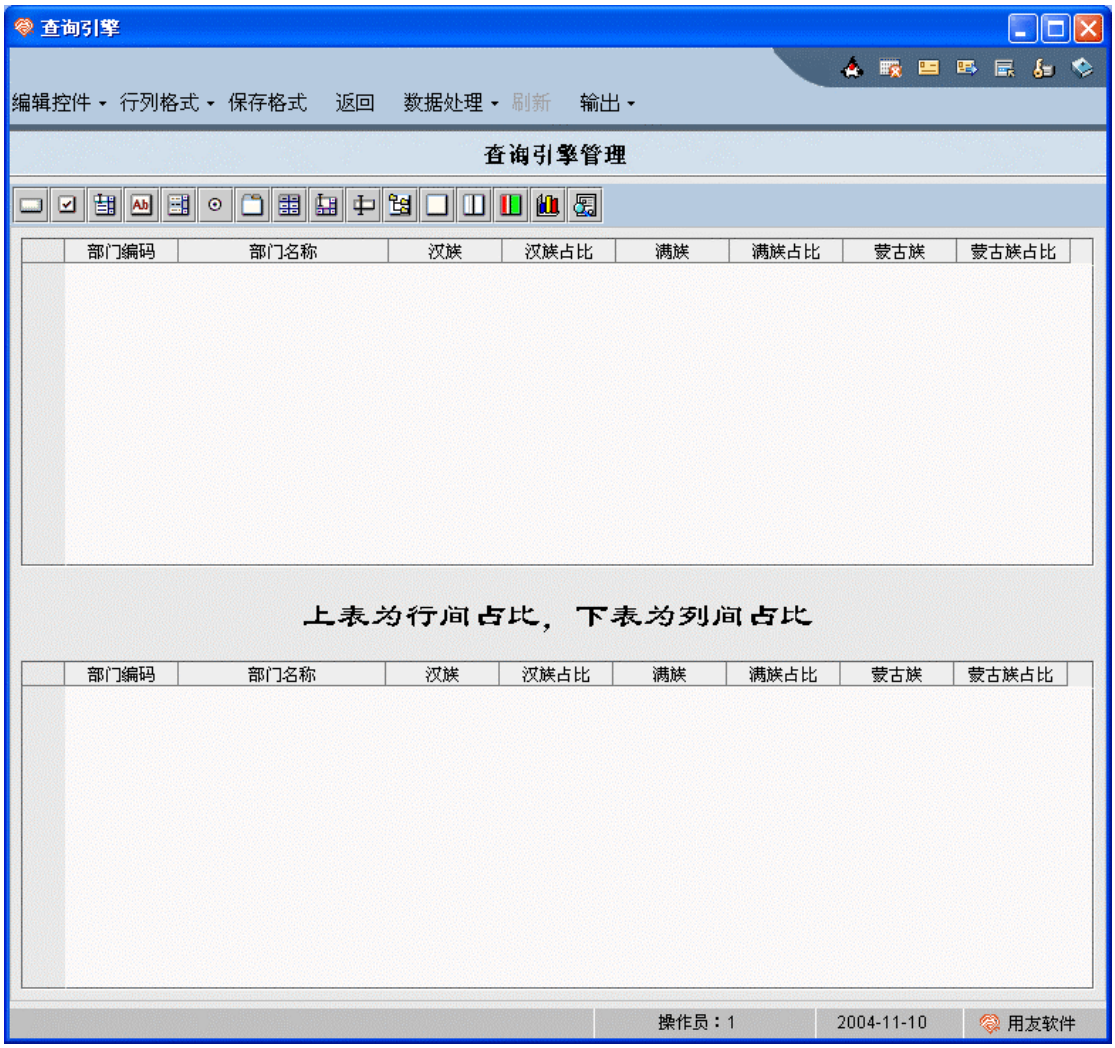


4、对“部门员工民族列间占比 A”进行数据加工，首先利用“获得结果集”向导生成从 percentColA 取数的代码，再利用“公式列”向导生成计算列间占比的代码——其实就是输入列间计算公式。有兴趣的读者可以考虑一下，如何利用复合查询获得列间占比效果；





5、创建格式对象，编码为 percentA，名称为“部门员工民族占比表 A”。引用查询“部门员工民族行间占比 A”和“部门员工民族列间占比 A”，在格式设计的南北位置放置两个表格分别绑定引用的查询，中部放一个标签作为提示。定义和引用百分比列格式，设置合计列；


查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
查询引擎管理
部门编码
部门名称
汉族
汉族占比
满族
满族占比
蒙古族
蒙古族占比
上表为行间占比，下表为列间占比
部门编码
部门名称
汉族
汉族占比
满族
满族占比
蒙古族
蒙古族占比
操作员：1
2004-11-10
用友软件

6、浏览报表。

	部门编码	部门名称	汉族	汉族占比	满族	满族占比	蒙古族	蒙古族占比
10	0305	ERP供应链产品部	11	9.73%	0	0%	2	18.18%
11	0307	ERP生产制造产品部	10	8.85%	0	0%	0	0%
12	0312	ERP服务支持部	7	6.19%	0	0%	1	9.09%
13	05	HR事业部	1	0.88%	0	0%	0	0%
14	0501	HR产品开发部	20	17.7%	3	37.5%	0	0%
15	0502	HR产品市场部	4	3.54%	0	0%	0	0%
16	0503	HR服务支持部	5	4.42%	0	0%	0	0%
17	aa	商务部	0	0%	2	25%	0	0%
18	bb	维修部	0	0%	1	12.5%	0	0%
19	合计		113		8		11	

	部门编码	部门名称	汉族	汉族占比	满族	满族占比	蒙古族	蒙古族占比
10	0305	ERP供应链产品部	11	84.62%	0	0%	2	15.38%
11	0307	ERP生产制造产品部	10	100%	0	0%	0	0%
12	0312	ERP服务支持部	7	87.5%	0	0%	1	12.5%
13	05	HR事业部	1	100%	0	0%	0	0%
14	0501	HR产品开发部	20	86.96%	3	13.04%	0	0%
15	0502	HR产品市场部	4	100%	0	0%	0	0%
16	0503	HR服务支持部	5	100%	0	0%	0	0%
17	aa	商务部	0	0%	2	100%	0	0%
18	bb	维修部	0	0%	1	100%	0	0%
19	合计		113		8		11	

5.3 投影类占比报表（投影占比）

【概述】

与上一模式的差别在于，本模式用于固定行固定列报表，上一模式用于动态行固定列报表。

【应用场景】

同上一模式，尤其常见于人事统计报表。

【适用性和不适用性】

适用于：

- 计算任何两个固定坐标单元之间数值比例；

不适用于：

- 动态数据之间的比例；

【解决方案】

采用数据加工提供的单元公式算法计算占比，公式中包含的行、列位置都是绝对坐标。

【开发步骤】

第一步：创建查询对象，利用向导设计定义投影交叉查询，并预留空列待占比结果填充；

第二步：在数据加工中获得上述结果集，利用单元公式算法描述占比列各单元格的公式；

第三步：创建格式对象，引用查询，设计格式。占比列可绑定百分比列格式；

第四步：浏览报表。

【范例】

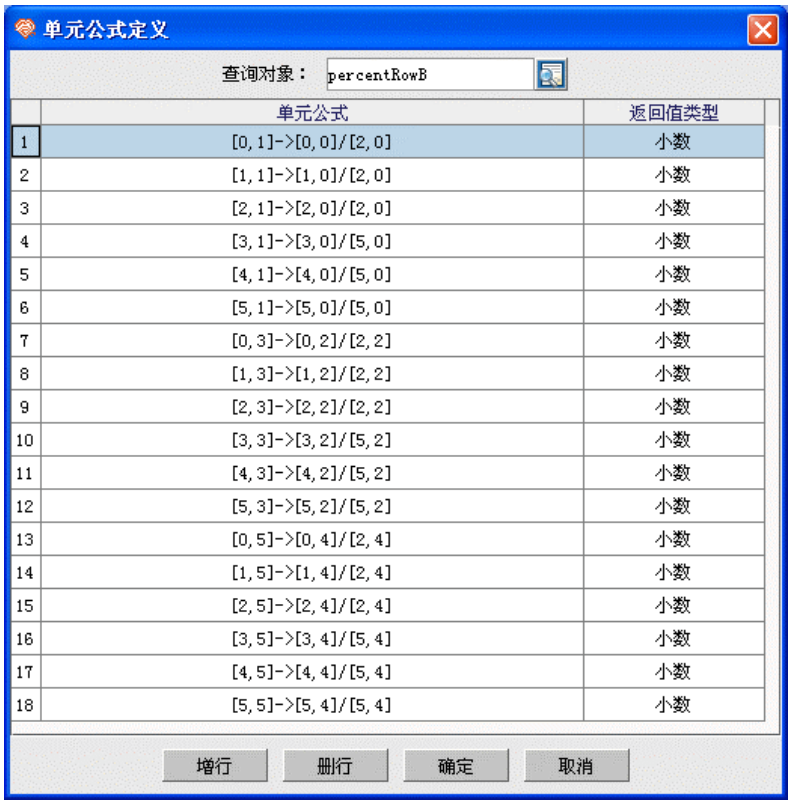
部门员工民族占比表（固定行列），表样如下，要求利用两张表格分别统计行间和列间占比。

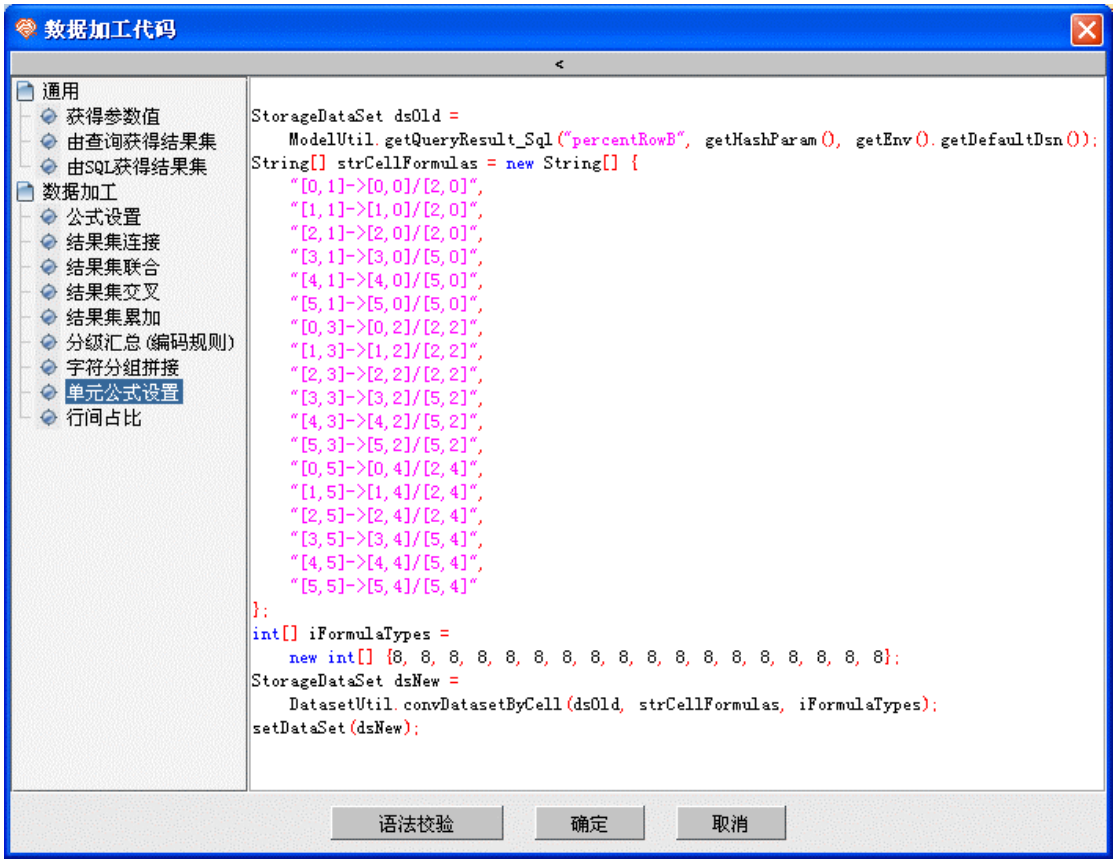
		汉族 人数	汉族 占比	满族 人数	满族 占比	蒙族 人数	蒙族 占比
直属部门	财务部						
	行政部						
	小计						
开发部门	HR 产品开发						
	HR 产品市场						
	小计						
合计							

1、 复制查询对象“部门员工民族统计 D”，修改编码为 percentRowB，名称为“部门员工民族行间占比 B”。修改向导设计的交叉属性页签，增加三个预留列（汉族占比、满族占比、蒙古族占比，条件均为空），准备用于填充加工出来的占比值；

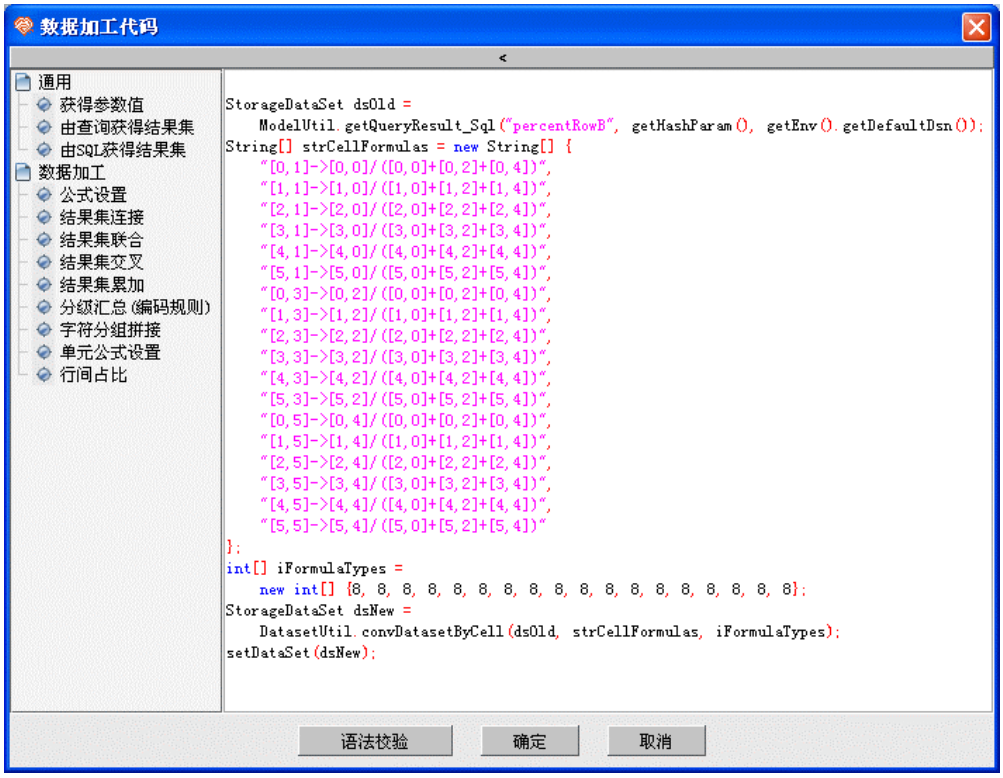
2、 对“部门员工民族行间占比 B”进行数据加工，首先利用“获得结果集”向导生成从 percentRowB 取数的代码，再利用“单元公式设置”向导输入描述行间占比的公式，注

意公式中采用[i,j]来代表第 i 行第 j 列的取值（i 和 j 从 0 排起）。由于本方案是对固定报表单元公式做的初步尝试，使用不便处敬请谅解；

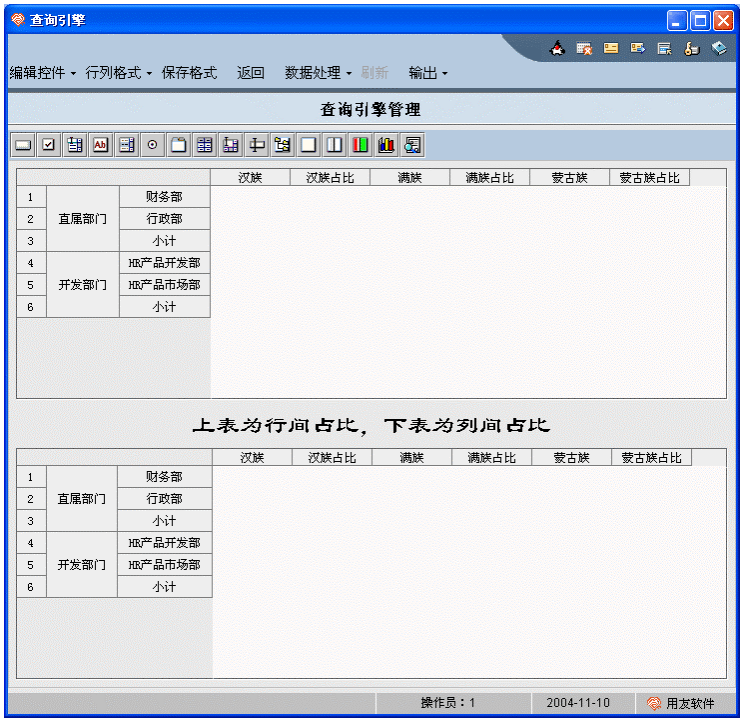




3、 复制前面定义好的查询对象“部门员工民族行间占比 B”，修改编码为 percentColB，名称为“部门员工民族列间占比 B”。修改数据加工代码如下；



4、 创建格式对象，编码为 percentB，名称为“部门员工民族占比表 B”。引用查询“部门员工民族行间占比 B”和“部门员工民族列间占比 B”，在格式设计的南北位置放置两个表格分别绑定引用的查询，中部放提示标签。定义和引用百分比列格式，做行头设计；



5、浏览报表。

			汉族	汉族占比	满族	满族占比	蒙古族	蒙古族占比
1	直属部门	财务部	5	45.45%	0		2	66.67%
2		行政部	6	54.55%	0		1	33.33%
3		小计	11	100%	0		3	100%
4	开发部门	HR产品开发部	20	83.33%	3	100%	0	
5		HR产品市场部	4	16.67%	0	0%	0	
6		小计	24	100%	3	100%	0	

			汉族	汉族占比	满族	满族占比	蒙古族	蒙古族占比
1	直属部门	财务部	5	71.43%	0	0%	2	28.57%
2		行政部	6	85.71%	0	0%	1	14.29%
3		小计	11	78.57%	0	0%	3	21.43%
4	开发部门	HR产品开发部	20	86.96%	3	13.04%	0	0%
5		HR产品市场部	4	100%	0	0%	0	0%
6		小计	24	88.89%	3	11.11%	0	0%

5.4 程序送数型报表

【概述】

报表的数据来源太过复杂，或者数据库里根本没有，只能通过开发人员自己编码获得数据。此时数据加工支持直接调用业务组编写的代码获得数据集，或者直接在加工窗口编码构造数据集，并由查询引擎做进一步的展现。

【应用场景】

开发人员根据参数设置结果，编写任意复杂的程序方法构造报表数据，该方法返回约定的数据结构。

【适用性和不适用性】

适用于：

- 完全利用字符串形式的参数取值就足以确定取数条件的报表；

不适用于：

- 报表取数方法中需要反复进行前后台交互；

【解决方案】

在数据加工代码中调用开发人员自己编写的服务器端代码，调用方法可以获得用户设置的参数取值，该方法返回的数据结构需要被转换成 StorageDataSet 之后才能被查询引擎识别。

【开发步骤】

第一步：开发人员编写一个服务器端方法用于提供数据，比如“StorageDataSet nc.bs.pub.TestBO.getData(Hashtable hashParam)”。如果开发人员必须使用另外的数据载体，比如输入查询条件 VO 数组和输出循环访问 VO 数组，那么就需要进行数据结构转换；

第二步：创建查询对象，定义参数，如果能够预先确定加工后列结构的话，可以利用手工设计定义一个假查询，用于描述列信息，便于格式设计；

第三步：在数据加工中调用开发人员自己编写的送数方法（使用 new BO 方式调用），传入参数哈希表，获得返回值，最后利用 setDataSet(StorageDataSet)设置待展现的数据集；

第四步：创建格式对象，引用查询，设计格式；

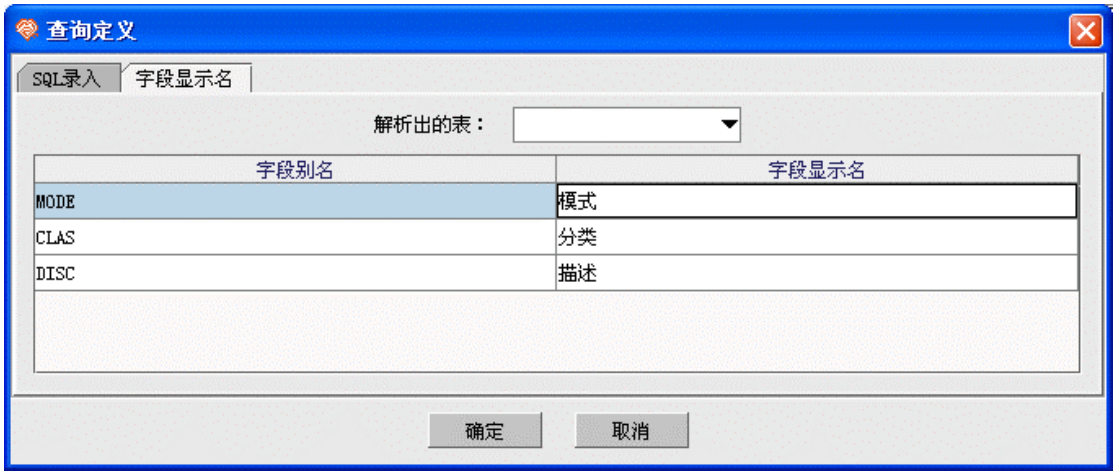
第五步：浏览报表。

【范例】

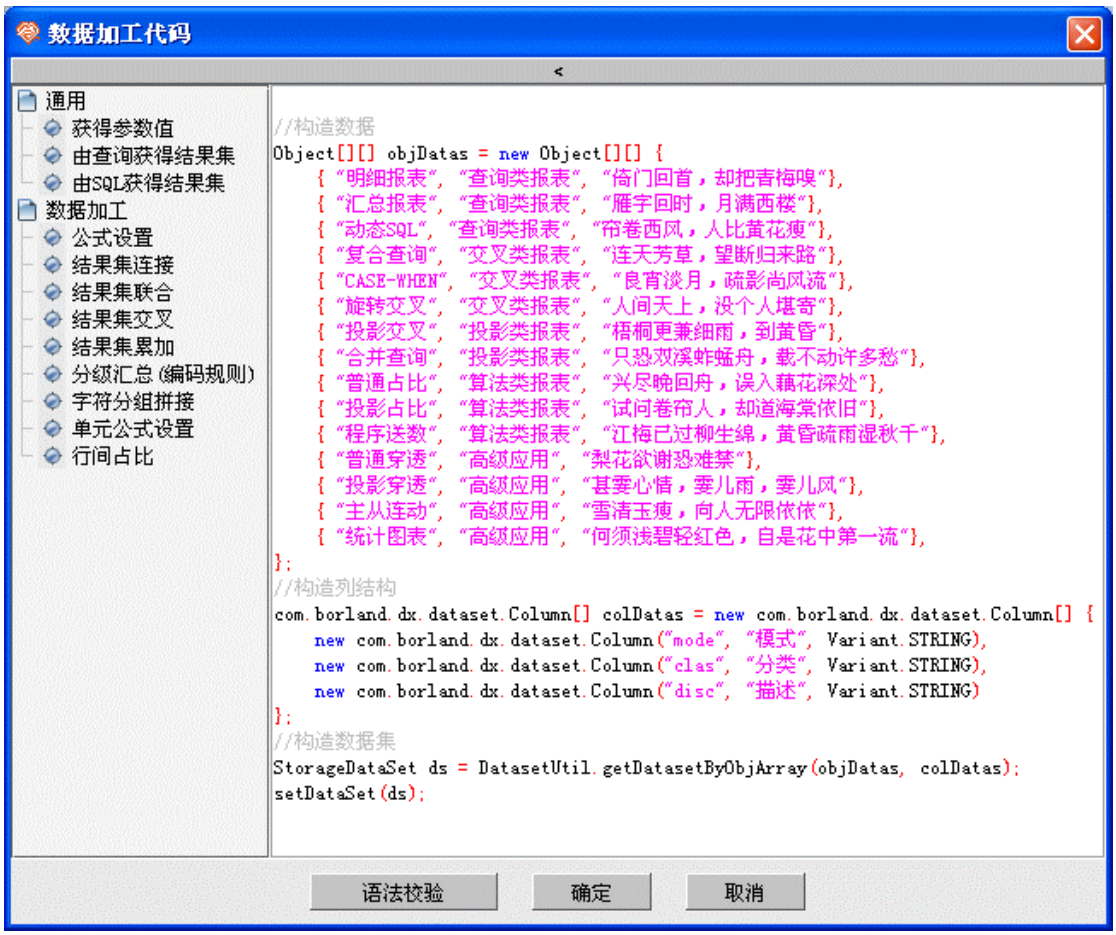
程序送数表，该报表要求展现本手册编写的查询引擎所有开发模式的信息——显然这些数据不可能从数据库中获得！既然需求如此，那么我们也只有自力更生写程序送数了。表样如下：

模式	分类	描述
明细报表	查询类报表	……
汇总报表	查询类报表	……
……	……	……

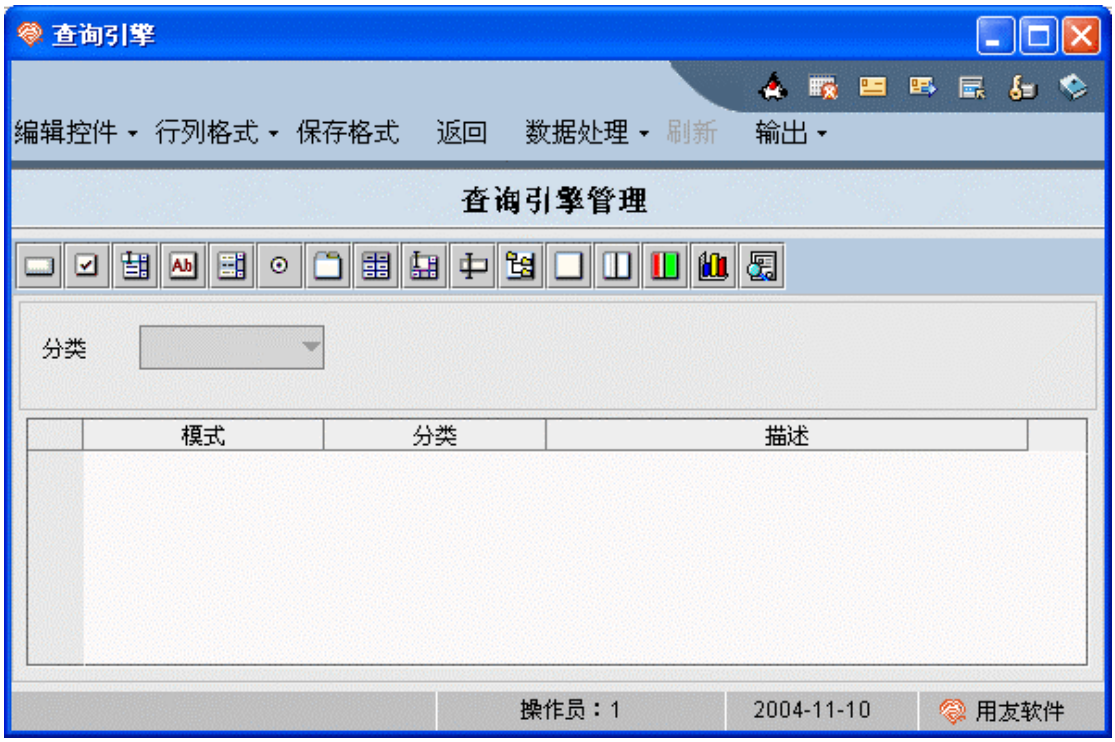
1、 创建查询对象，编码为 progdata，名称为“程序送数”。利用手工 SQL 设计定义假查询（注意“select 常量”的语法在 Oracle 下可能有问题，届时我们加一个“from 某张表”就可以了，因为我们的目的不是查出结果，而只是提供与加工后结果相匹配的列结构）；



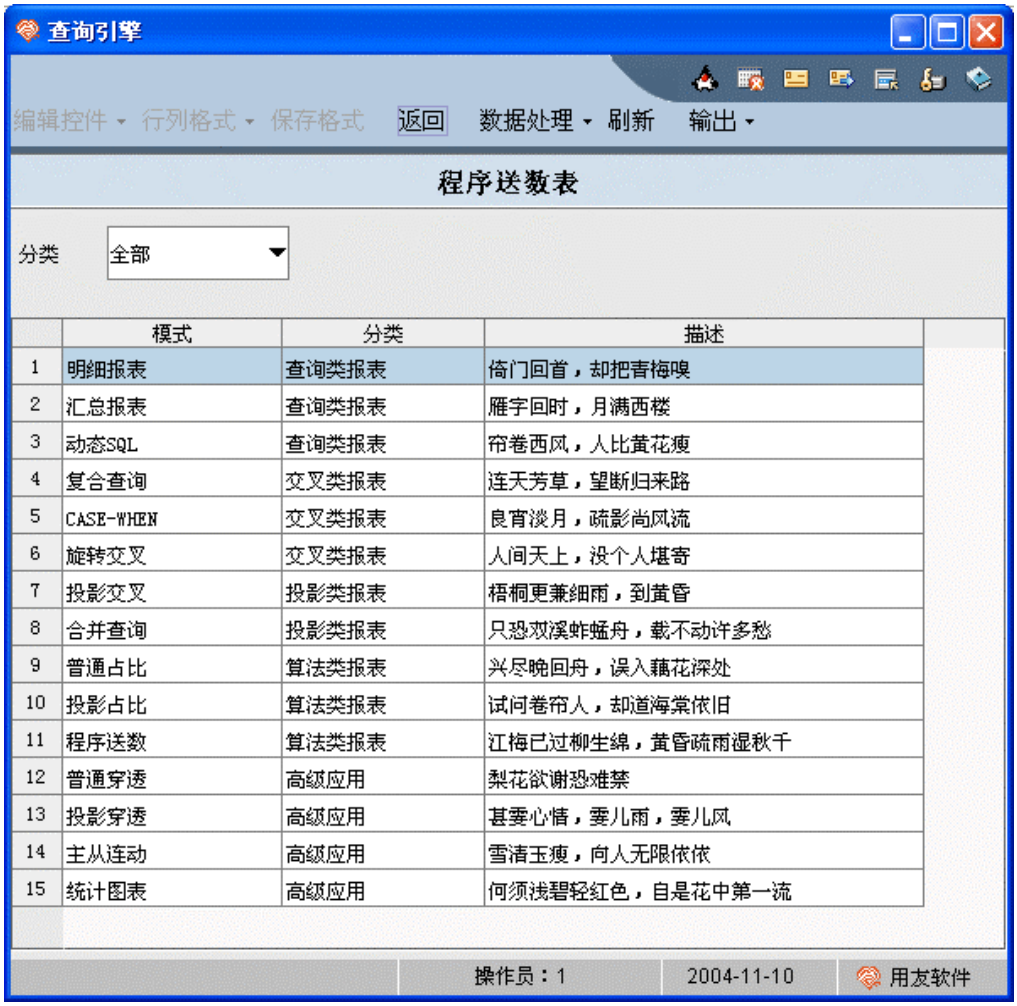
2、 自由编写加工代码，最后通过 setDataSet()送数。这里出于演示环境需要直接在编码窗进行了编程，但我们还是建议大家把加工代码封装成一个调用方法来做；



3、 创建格式对象，编码为 progdata，名称为“程序送数表”。引用查询“程序送数”，在格式设计中放置表格控件，绑定引用查询；


查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
查询引擎管理
分类
模式
分类
描述
操作员：1
2004-11-10
用友软件

4、报表浏览。


查询引擎
编辑控件
行列格式
保存格式
返回
数据处理
刷新
输出
程序送数表
分类
全部
模式
分类
描述
1 明细报表 查询类报表 倚门回首，却把青梅嗅
2 汇总报表 查询类报表 雁字回时，月满西楼
3 动态SQL 查询类报表 帘卷西风，人比黄花瘦
4 复合查询 交叉类报表 连天芳草，望断归来路
5 CASE-WHEN 交叉类报表 良宵淡月，疏影尚风流
6 旋转交叉 交叉类报表 人间天上，没个人堪寄
7 投影交叉 投影类报表 梧桐更兼细雨，到黄昏
8 合并查询 投影类报表 只恐双溪蚱蜢舟，载不动许多愁
9 普通占比 算法类报表 兴尽晚回舟，误入藕花深处
10 投影占比 算法类报表 试问卷帘人，却道海棠依旧
11 程序送数 算法类报表 江梅已过柳生绵，黄昏疏雨湿秋千
12 普通穿透 高级应用 梨花欲谢恐难禁
13 投影穿透 高级应用 甚要心情，要儿雨，要儿风
14 主从连动 高级应用 雪清玉瘦，向人无限依依
15 统计图表 高级应用 何须浅碧轻红色，自是花中第一流
操作员：1
2004-11-10
用友软件

6. 高级应用

以下介绍的支持穿透、连动和统计图的报表是在已经完成基本报表的前提下，根据用户在数据分析、数据挖掘、领导查询和辅助决策方面的需求进行的高级开发应用。

6.1 支持穿透的非投影交叉类报表（普通穿透）

【概述】

根据某张报表当前选中区域内的数据，切换到另一张报表，而前者选中数据和后者显示数据之间具有潜在的业务联系。一般来说，穿透多用于从汇总报表联查明细报表，比如在汇总的费用报表当中选中某个月份，然后穿透到该月份每天的费用明细情况。此类应用在领导查询中具有十分关键的作用，但本模式的解决方案涉及简单的代码片段编写，仅建议熟悉技术的开发人员使用。

【应用场景】

用户报表除基本的查询、数据处理和输出的需求外，还要求基于选中行根据指定的业务规则穿透到另一张报表与该行数据相关的明细数据，且此穿透过程可不断延续。

【适用性和不适用性】

适用于：

- 根据选中行数据进行任意自定义规则的穿透；

不适用于：

- 多行穿透；

【解决方案】

查询引擎提供的穿透解决方案包括两个环节：定义者在设计态描述穿透规则，浏览者在运行态根据规则执行穿透。后者属于领导或业务人员的低技能操作，不再赘述，这里只介绍定义穿透的相关内容。

穿透是从查询到查询之间进行的，所以首先必须明确哪个是源查询，哪个是目标查询。目标查询必须创建一个（或多个）接收参数，接收参数是穿透双方的通讯纽带，目标查询模型必须能够根据这个参数过滤出所需要的数据。源查询中需要创建一个（或多个）穿透规则，规则中需要描述以下情节：

1、 从源查询的选中行中获得哪些信息；
2、 目标查询是谁；
3、 把获得的选中信息经过怎样的处理再送给目标查询的接收参数；

上述情节需要通过 JAVA 代码片段描述，但对于简单的单值穿透，可以利用规则定义界面左侧的代码向导树直接生成代码片段。代码中的常用接口方法包括：

1、 String getPeneValue(String colName); //获得选中行列别名为 colName 的单元取值；
2、 Hashtable getHashParam(); //获得当前查询参数设置的哈希表

【开发步骤】

第一步：分析穿透前后的数据，并找出可以作为其间联系纽带的接收参数；

第二步：定义源查询和目标查询及其对应的报表（具体实现与本模式无关）；

第三步：在目标查询参数定义中创建接收参数，在 SQL 向导设计、SQL 整理或者数据加工中指定根据接收参数如何过滤静态查询查出来的数据；

第四步：在源查询中利用“查询间穿透”向导定义穿透规则，需描述的信息参见解决方案；

第五步：浏览源查询对应的报表，选中行，利用表头右键菜单提供的穿透功能选择规则执行穿透，检查穿透前后的数据是否符合用户的需求逻辑；

【FAQ】

1、 为什么我定义的穿透执行后穿出的数据不对？
答：请检查以下两个地方：（1）目标查询是否根据穿透参数进行了正确过滤；（2）穿透规则是否把选中数据传给了正确的接收参数。

【范例】

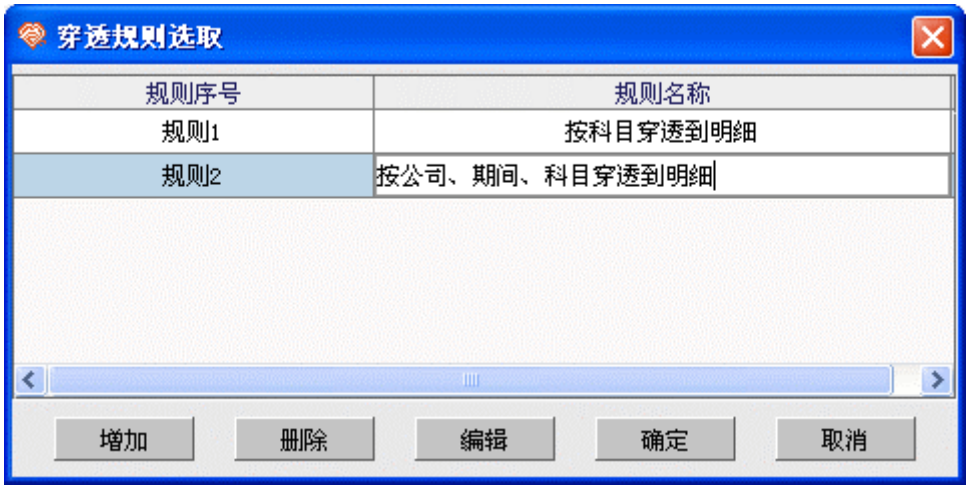
费用汇总表穿透到费用明细表（这两张表如本章第二节所定义），要求定义两种穿透规则，第一种根据选中行的科目穿透到该科目下的所有费用明细，第二种根据选中行的公司、期间、科目穿透到相应明细。两种穿透均要求保证浏览汇总表时待定条件取值的传递。

需要说明的是：对于一般联查明细应用而言，本范例中的第二种穿透规则更具有其实际意义，这里介绍第一种穿透规则（单值穿透）的目的无非是想让读者循序渐进地理解第二种穿透规则（多值穿透）的定义原理。

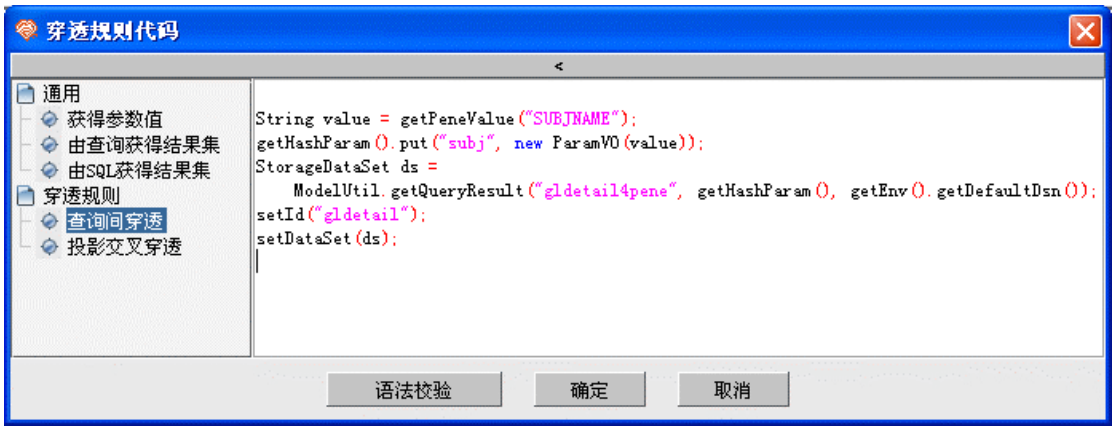
1、 复制查询对象 gldetail，修改编码为 gldetail4pene，名称不变，作为目标查询。在参数定义中增加公司和科目两个参数，它们将与已有的两个期间参数共同作为接收参数：

2、 修改该对象的筛选条件，增加两个待定条件，用于根据公司和科目进行过滤；

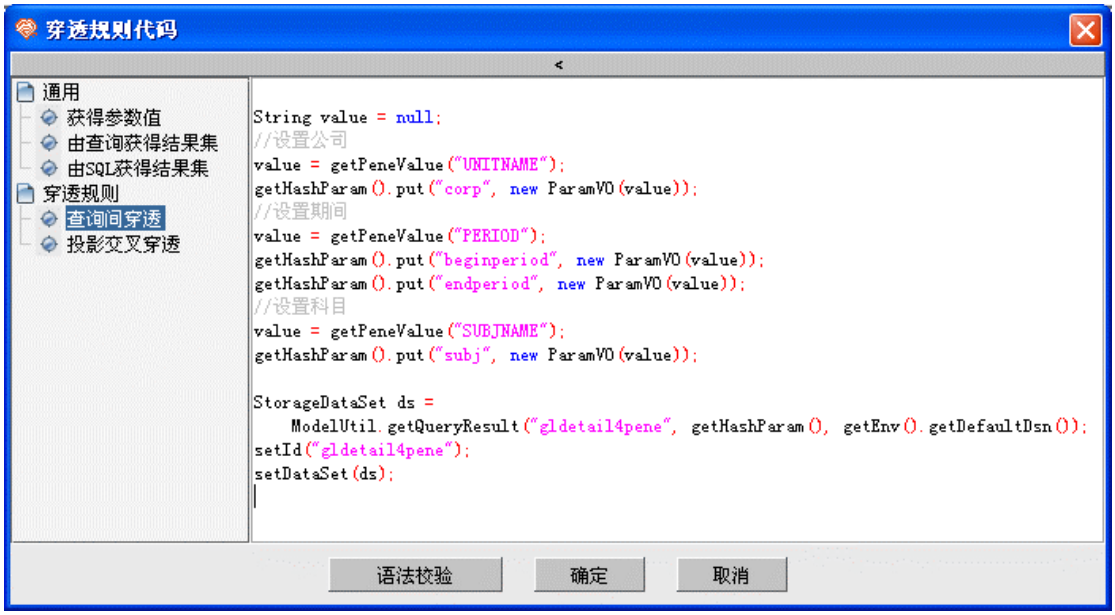
3、复制查询对象 glsum，修改编码为 glsum4pene，名称不变，作为源查询。创建两条穿透规则，分别名为“按科目穿透到明细”和“按公司、期间、科目穿透到明细”；



4、编辑该对象的第一条穿透规则，明确目标：我们要根据选中行的科目字段（在 SQL 设计中的别名为 subjname）取值穿透到费用汇总查询（编码 gldetail2pene），其接收参数为科目（参数编码 subj）。利用左侧的“查询间穿透”代码生成向导可以轻松生成规则；



5、编辑该对象的第二条穿透规则，明确目标：我们要根据选中行的公司（unitname）、期间（period）和科目（subjname）字段取值穿透到费用汇总查询（gldetail2pene），其接收参数为公司（corp）、期间（beginperiod，endperiod）和科目（subj）。利用左侧的“查询间穿透”代码生成向导生成规则片段后（或直接复制第一条规则的代码）稍加修改，把从选中行获得的公司、期间、科目信息分别送给参数哈希表的对应参数键值即可；



6、创建格式对象，编码为 penetrationA，名称为“费用汇总穿透”；

7、引用查询“费用汇总穿透”；

8、进行格式设计：添加表格控件，绑定数据集费用汇总穿透；

9、浏览报表（先输入参数），选中报表表体某行；

	公司名称	会计期间	科目名称	本币借发生额	本币贷发生额
98	常州分公司	02	固定资产购入	0.0000	16696.5879
99	常州分公司	02	固定资产减值准备	0.0000	3900.0000
100	常州分公司	02	固定资产清理	9852.6639	0.0000
101	常州分公司	02	管理费用	4200.0000	0.0000
102	常州分公司	02	进项税额	9696.8000	0.0000
103	常州分公司	02	库存商品	-1030.0000	0.0000
104	常州分公司	02	累计折旧	4511.0600	7329.5600
105	常州分公司	02	物资采购	5.7040	0.0000
106	常州分公司	02	银行存款	0.0000	4200.0000
107	常州分公司	02	应付账款	0.0000	6.6737

10、根据选中行按照第一种规则进行穿透，注意穿透结果的科目列（科目=累计折旧），以及初始参数设置取值（年度=2004，期间=01 至 03）的传递；

11、回退到穿透前状态，根据选中行按照第二种规则进行穿透，注意穿透结果的公司、期间和科目列（公司=常州分公司，期间=02，科目=累计折旧）；

6.2 支持穿透的投影交叉类报表（投影穿透）

【概述】

本类报表与前一类报表有两点不同在于：

1、 前一类是根据选中行进行穿透，本类是根据选中单元进行穿透；

2、 前一类传递的信息是值，即选中行某列（或某几列）的取值，本类传递的信息是 WHERE 条件，即选中单元对应的行条件和列条件。

由于投影交叉表的单元数据都是统计（汇总或计数）结果，因此穿透的目标查询往往是与这个统计结果对应的明细结果。

【应用场景】

人事报表的反查是本类报表的最常见应用。比如由列条件“学历='本科'”和行条件“职务='处长'”确定的表单元中的人数，穿透到所有具有上述学历和职务条件的人员明细表。

【适用性和不适用性】

适用于：

- 投影交叉查询的穿透；

不适用于：

- 合并查询的穿透；
- 多单元格的穿透；

【解决方案】

目标查询提供一个接收参数，这个参数的取值就是一个 where 条件字符串，目标查询利用 SQL 整理把 where 串追加到目标查询定义的静态 SQL 后面。另一方面，源查询定义一个穿透规则，这个规则获得选中单元对应的行列条件，然后传递给目标查询的接收参数，从而实现相同筛选条件下统计值到明细值的穿透。

【开发步骤】

第一步：定义源查询和目标查询及其对应的报表（具体实现与本模式无关）；

第二步：在目标查询参数定义中创建接收参数，利用 SQL 整理把接收参数的取值追加到目标查询静态 SQL 的筛选条件当中；

第三步：在源查询中借助“投影交叉穿透”向导定义穿透规则；

第四步：浏览源查询对应的报表，选中表单元，利用表头右键菜单提供的穿透功能选择规则执行穿透，检查穿透前后的数据是否符合用户的需求逻辑；

【FAQ】

1、在运行态作动态投影交叉之后，穿透规则是否仍旧适用？
答：适用。

【范例】

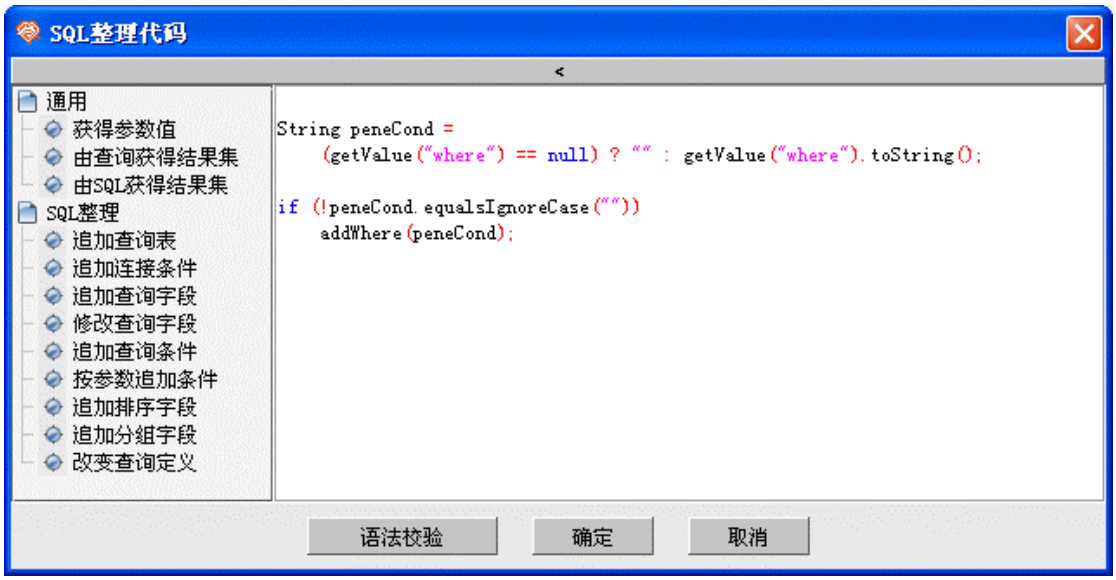
从部门员工民族人数统计表（投影交叉）穿透到部门员工民族明细表。

		汉族	满族	蒙古族
直属部门	财务部			
	行政部			
	小计			
开发部门	HR 产品开发部			
	HR 产品市场部			
	小计			
合计				

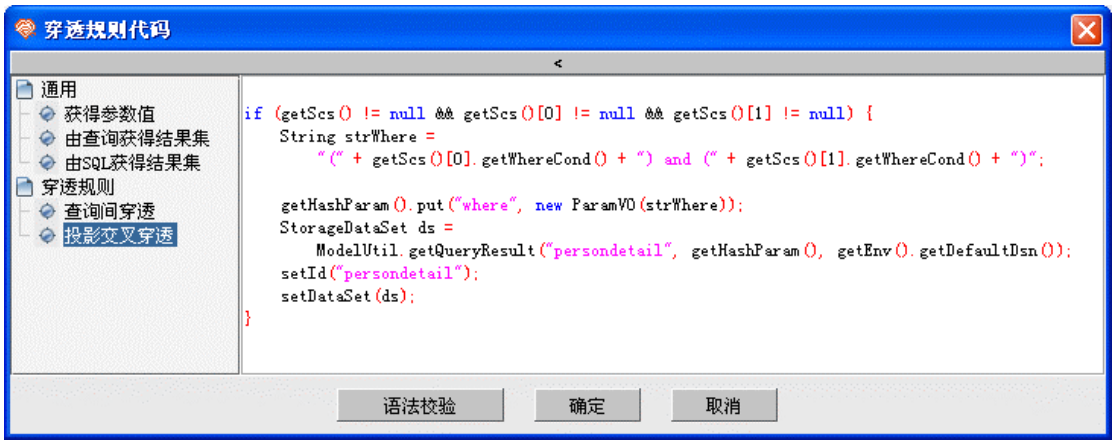
部门编码	部门名称	姓名	性别	民族	出生日期

1、创建查询对象，编码为 persondetail，名称为部门员工民族明细。SQL 设计中联查部门档案、人员档案和人员辅助信息，获得各部门的人员明细信息；

2、创建接收参数 where，通过 SQL 整理把 where 的取值追加到 SQL 定义中的筛选条件中；



3、复制查询对象“部门员工民族统计 D”，修改编码为 personcount，名称为部门员工民族人数。定义穿透规则“穿透到人员明细”，利用“投影交叉穿透”编码向导，输入目标查询（部门员工明细）和接收参数（where），由系统生成穿透代码；

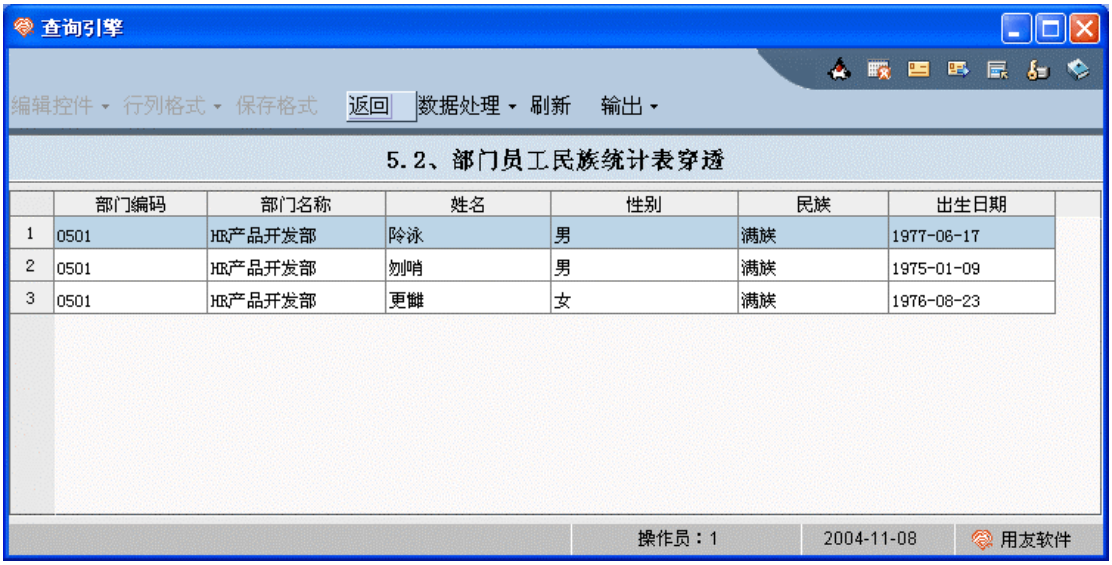


4、创建格式对象，编码为 penetrationB，名称为“部门员工民族统计表穿透”。引用查询“部门员工民族人数”；

5、进行格式设计：添加表格控件，绑定数据集费用部门员工民族人数；

6、报表浏览，选中某单元格（HR 产品开发部和满族的交叉点：3 人）；

7、根据选中单元格按照第一种规则进行穿透，注意穿透结果的特性（共 3 行，部门=HR 产品开发部，民族=满族）；



6.3 支持主从连动的报表

【概述】

主表与子表之间根据关联字段进行连动。主表数据选中一行时，子表数据自动显示与该行对应的记录。

【应用场景】

用户更侧重于从界面获得动态分析效果，利用多表展现浏览业务数据的级联关系。

【适用性和不适用性】

适用于：

- 主表与子表绑定的数据集之间存在关联字段；

不适用于：

- 多关联字段连动（V3 不支持，最新开发版本已支持）；

【解决方案】

在引用查询的主从设置中指定引用数据集之间的关联字段，此后绑定相关数据集的控件（目前是表格和图表）之间就建立了连动关系。

由于关联字段通常是外键，一般不希望显示在表格上，因此可以将其设为隐藏列。

【开发步骤】

第一步：定义主查询和子查询及其对应的报表（具体实现与本模式无关）；

第二步：创建格式对象，在引用查询的主从设置中设定引用数据集之间的主从关系；

第三步：进行格式设计，添加表格（或图表）控件，绑定主数据集和子数据集。最常见的情况是界面上放两个上下并排或左右并排的表格，一个做主表，一个做子表；

第四步：浏览报表，选中主表的数据行，检查子表数据是否随之动态变化；

【FAQ】

1、是否需要用主子表控件展现连动？

答：该控件已不建议使用。建议的方法是将利用表格或图表绑定主数据集和子数据集，然后通过主从规则设置其关联字段。

【范例】

部门信息与人员信息连动报表——由两个表格组成，主表为部门信息，子表为人员信息，主表选中某部门的时候子表显示该部门的人员。

部门编码	部门名称

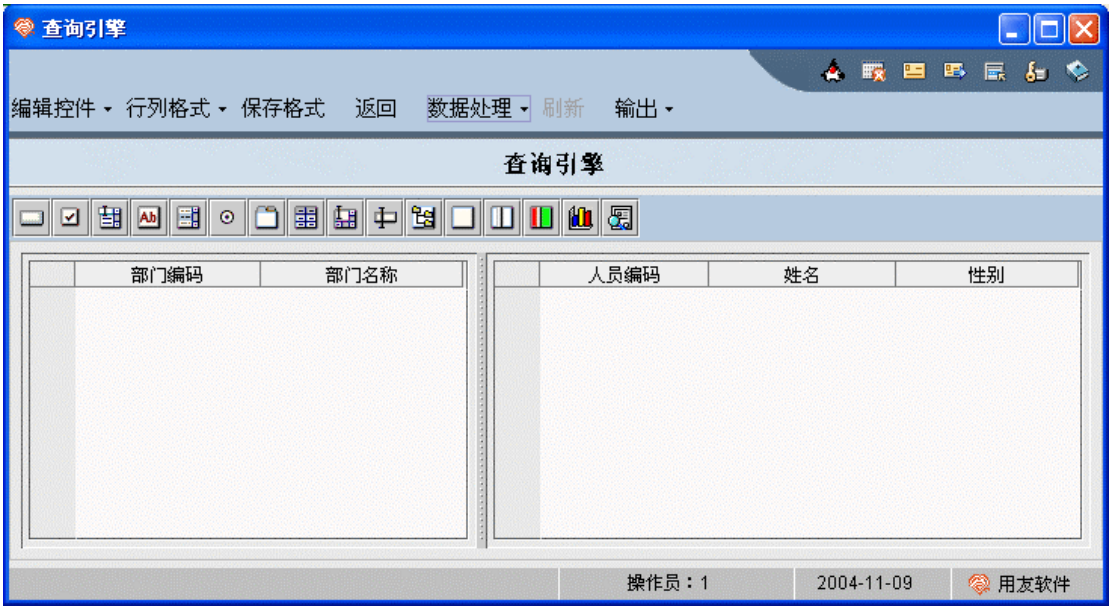
人员编码	人员名称	性别

1、创建查询对象，编码为 deptinfo，名称为“部门信息”。做简单单表查询获得部门明细信息，查询字段包含部门主键（此字段将作为主从联动的纽带）；

2、创建查询对象，编码为 psninfo，名称为“人员信息”。做简单联查获得人员明细信息，查询字段包含所属部门（外键）；

3、创建格式对象，编码为 masterdetailA，名称为“部门员工信息连动”。引用查询“部门信息”和“人员信息”，在主从设置中指定两查询的部门主键作为连接字段；

4、进行格式设计：添加拆分窗格控件，左右面板均放入表格控件，分别绑定数据集部门信息和人员信息，并在它们的列格式中均指定部门主键列为不显示；



5、浏览报表，选中左表某部门的时候，右表将显示该部门下的人员；

6.4 支持统计图表的报表

【概述】

随着企业信息化的发展，对图文并茂的文档型报表的需求与日俱增，用户除了要看到传统意义上的报表数据以外，还希望通过更为直观的统计图展现来评估企业绩效。

【应用场景】

表样为纯图表或表图结合的方式，CHART 部分基于业务数据提供分组统计的直方图、折线图、饼图等分析展现，或指针预警形式的仪表盘展现。

【适用性和不适用性】

适用于：

- 以某个列作为分组列统计另外若干数值列的汇总结果；

不适用于：

- 同列数据的行间比较；

【解决方案】

查询引擎提供封装好的图表控件，可放置与表单的任何位置。每个统计图绑定一个数据集，然后根据数据集设定图表类型、展现方式、分组列和统计列信息。

查询引擎 V3 支持包括分类图、饼图和仪表图在内三大类共十余种常用统计图。其中分类图

用于分序列统计，有 2D 和 3D 两种，可以选择是水平方向或垂直方向展开，如果选择了各系列单独设置图例，还可以对不同的系列设置不同的展现类型。饼图是常见的汇总图表。仪表图将每个系列的数据在一个仪表盘中显示，用于管理决策者监督数据处于正常状态或临界预警状态。仪表图一次展现数据中的一条记录，可以通过仪表图中的导航工具栏浏览数据。

【开发步骤】

第一步：定义查询及其对应的报表（具体实现与本模式无关）；

第二步：创建格式对象，引用查询；

第三步：进行格式设计，添加图表控件，双击控件绑定数据集，利用右键菜单的数据绑定功能设置图表类型和分组统计信息；

第四步：浏览报表，绑定相同数据集的图表与表格之间存在主从连动关系；

【FAQ】

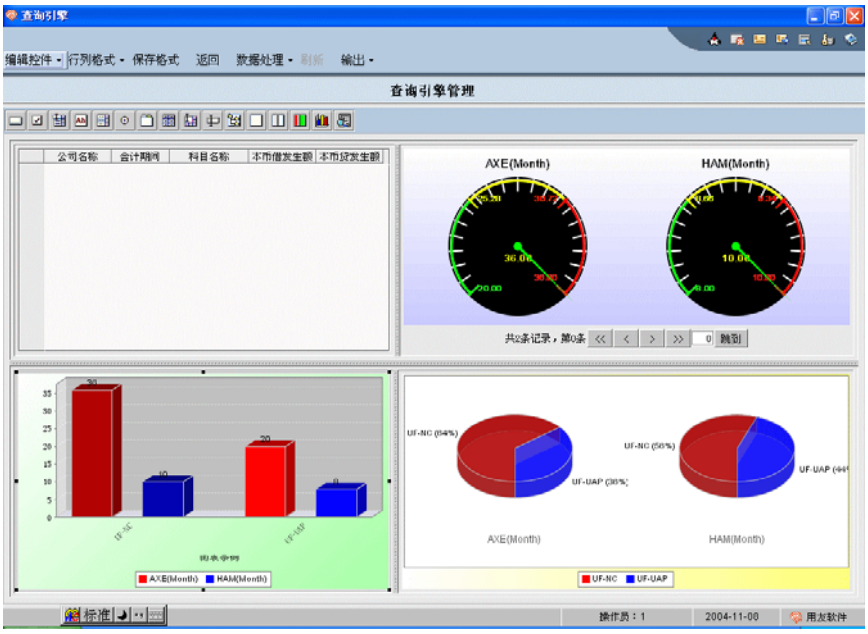
1、能否展现数据交叉后对应的图表？
答：在运行态可以，设计态无法预设。

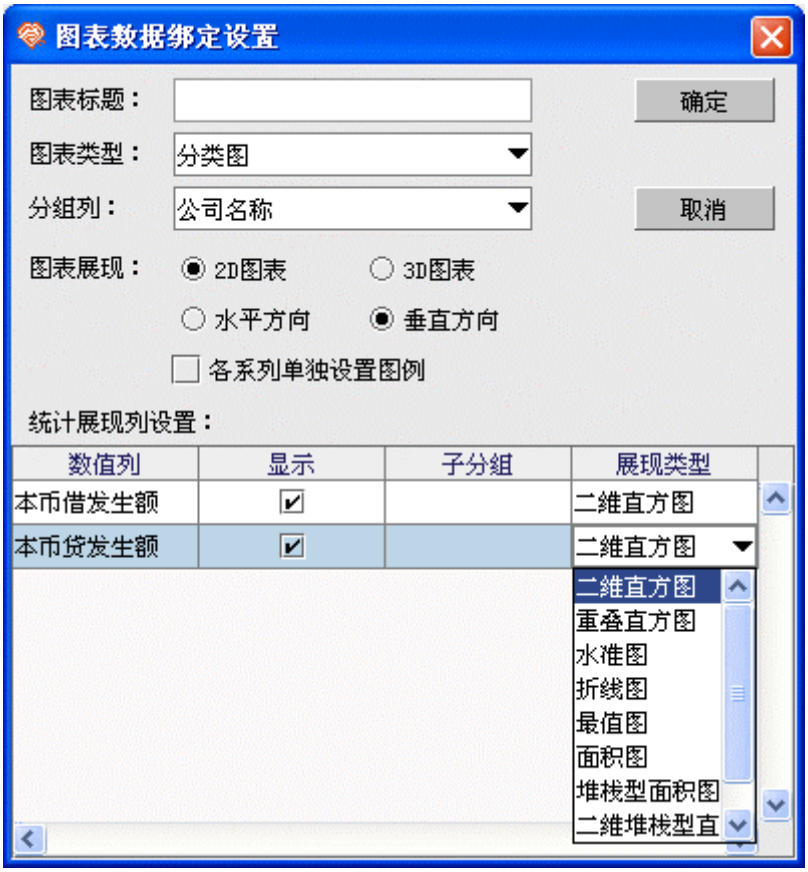
【范例】

费用汇总统计图报表——对模式 2.2 范例的费用汇总表进行统计图表分析。

1、创建格式对象，编码为 chartA，名称为“费用汇总统计图表”。引用查询 glsum4pene；

2、进行格式设计，先后放入三个拆分窗格（一个垂直拆分，两个水平拆分），在分隔出的四个面板中分别放入一个表格和三个图表控件，并均绑定数据集。通过图表控件右键菜单中的数据绑定功能做以下图表设置：第一个图表类型为仪表图，显示所有统计列；第二个图表类型为分类图，分组列为公司，显示所有统计列，展现类型为三维直方图；第三个图表类型为饼图，分组列为公司，显示所有统计列；





3、报表浏览，注意选中表格行或选中图表区域引起的同步连动效果；

4、由于界面上的四个控件绑定到同一个数据集，因此任何一种针对数据集的数据处理操作（查找，过滤，排序，公式）都将影响到四个控件的展现。比如我们通过过滤设置滤掉常州分公司以外的公司，则可以看到图示的同步连动效果；

第三章　高开发难度报表

1. 基于行业报表工具开发的报表

1.1 标准报表

- **简述**

为了满足快速开发的要求，NC 行业在 2003 年底总结了一种结合配置工具的报表开发方法，通过在实践中使用并不断优化该方法，对它进行推广的时机已经成熟。报表模板与单据模板的数据展现非常相似，以下面的报表为例：

	tabacco_code	tabacco_name	itemvalue
	卷烟编码	卷烟名称	数值
1	CHN31010001B000297000002000	84硬盒红双喜(11mg)	126049665.25
2	CHN31010001B100297000@@20@@	84硬盒红双喜(11mg出口)	7404948.04
3	CHN31010001B000297020002000	84硬盒红双喜(11mg特)	185090417.57
4	CHN31010007F000220040001000	84硬盒红双喜(12支装精品)	2.40
5	CHN31010001F000220040001000	84硬盒红双喜(15mg精品)	109796627.49
6	CHN31010001F000254000002000	84硬盒红双喜	2556260849.68
7	CHN31010001F100254000@@20@@	84硬盒红双喜(出口)	26825509.00

对于这三列，每列我们都要像单据模板一样定义其属性，包括列名、显示顺序、是否公式列等等。传统的报表开发步骤为：首先定义报表项目及其属性，包括名称、是否显示、公式等等；然后需要定义一个 VO 类，使其包括所有项目以存储其数据；最后需要创建 DMO 类，将所有数据库查询逻辑放入其中，利用它实例化 VO 对象。这样的步骤能不能简化呢？答案是肯定的。那么行业报表开发方法到底在哪里做了优化？

1) 报表模板定义的项目和 VO 类的定义实际是重复的，也就是说，我们只要定义了前者，就可以按一定规则创建后者。因此，VO 类只需要一个通用的类即可。具体的转换规则和相关细节会在后面介绍。
2) 对于大多数报表，通常的数据模型都是多个表关联查数。如果人工创建 DMO 类，其查询语句的书写会很麻烦。实际上，对于标准的多表连查，是可以使其查询语句自动生成的，只不过，我们需要配置表连接信息。行业方法不仅提供了直观的配置工具，还用一个机制保证了这种方案的可行性。因此，DMO 的创建也可以免去。
3) 要实现上面两点，行业方法需提供通用报表 VO，通用报表 DMO 等基础性组件，实际上，除此之外，行业方法还提供了多个报表基类，开发人员只需要继承相应的基类，写很少的代码即可快速完成报表。

- **应用场景**

1) 标准的报表：报表的每行都是由多个表连接查询出来的原始数据组成，即查询出来的每行数据严格地对应着报表的一行。

2)　分块填充的报表：这是标准报表常见的一个变体，我们将在随后一节专门介绍。

- **解决方案**

1)　报表模板定义和 VO 定义的对应关系

通用报表 VO 在内部实际维护着一个名称-值的 Map，因此它是动态的。创建报表 VO 是根据一些元信息，这些元信息被封装到一个类中，它叫 ReportMetaClass，而这些元信息来自于报表模板定义。ReportVOMetaClass 一方面相当于报表 VO 工厂，另一方面它对拼接查询语句起到了至关重要的作用，因此，我们需要重点介绍行业方法使用的一项重要规则。

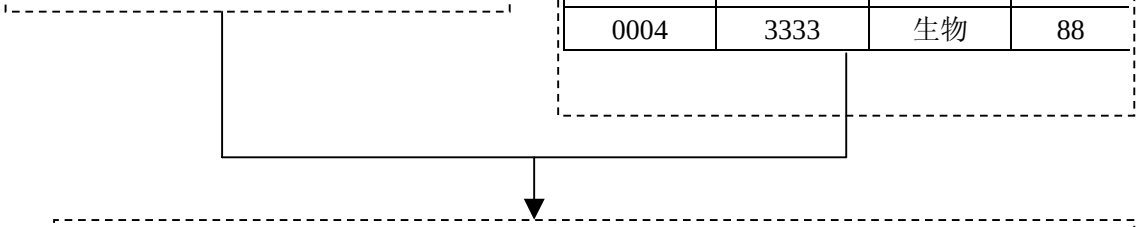
假设现在需要联合两个表制作报表，表结构如下：

学生主表 student

pk_student	name	grade
2222	Crist	1
3333	Merry	2

学生科目成绩 subject

pk_subject	pk_student	sub_name	mark
0001	2222	数学	80
0002	3333	英语	90
0003	2222	体育	98
0004	3333	生物	88



最终报表

学生姓名	学生年级	科目	成绩
Crist	1	数学	80
Crist	1	体育	98
Merry	2	英语	90
Merry	2	生物	88

对于这个报表，我们定义其报表模板为：

学生主键	学生姓名	学生年级	科目主键	主表主键	科目	成绩
a_pk_student	a_name	a_grade	b_pk_subject	b_pk_student	b_sub_name	b_mark
不显示	显示	显示	不显示	不显示	显示	显示

仔细观察对列名称的的定义，每项前面都含有”a_”或”b_”，其中，”a”是对 student 表起的别名，”b”是对 subject 表起的别名，前面提到的报表 VO 元信息类 ReportVOMetaClass 通过将第一个”_”符号翻译成”.”符号便可与数据库字段直接对应，如果我们为其设置表连接语句：student a left outer join subject b on a.. pk_student = b. pk_student，即可自动构造完整的查询语句，如下：

```
select a.pk_student, a.name, a..grade, b.pk_subject, b.pk_student, b.sub_name, b.mark
    from student a left outer join subject b on a.. pk_student = b. pk_student
```

利用统一的 DMO 将数据查询出来，并将报表 VO 示例化，在前台即可得到一组 VO 对象，因为它们的参数与报表模板定义保持着一致，所以可以正确显示报表数据。

2) 行业报表配置工具介绍

行业方法使用了一套强大的Excel工具对报表开发进行配置,它提供对报表模板的配置,还提供对查询模板的配置。下面将介绍这个工具，介绍中使用的例子会用到两张表，它们是主子表的关系，表结构为：

	Name	Code	Data Type
1	主键	pk_packcost	char(20)
2	公司主键	pk_corp	char(4)
3	备注	vmemo	varchar(128)
4	部门	pk_deptdoc	char(20)
5	材料大类	pk_materialclass	char(20)
6	材料档案	vinvmanid	char(20)
7	材料耗量	ninvmannum	decimal(20,8)
8	材料单价	ninvmanprice	decimal(20,8)
9	材料耗用金额	ninvmanmny	decimal(16,4)
10	年度	vyear	char(4)
11	季度	vquater	smallint
→	月度	vmonth	smallint

表 1

	Name	Code	Data Type
1	主键	pk_packcost_b	char(20)
2	公司主键	pk_corp	char(4)
3	分摊主表主键	pk_packcost	char(20)
4→	存货ID	pk_invmandoc	char(20)
5	月产量	nmonthoutputnum	decimal(20,8)
6	可分摊产量	nmayoutputnum	decimal(20,8)
7	分摊产量	noutputnum	decimal(20,8)
8	单价	nprice	decimal(20,8)
9	分摊金额	nwastemny	decimal(16,4)
10	分摊耗量	nwastenum	decimal(20,8)
11	月单耗量	nmouthwastenum	decimal(20,8)
→	月单耗金额	nmouthwastemny	decimal(16,4)

表 2

配置工具基于 Excel，我们的配置步骤为：

A) 配置基本信息

a) 在配置页签中填入下面的信息：它表示我们使用的表个数为 2 个，行业代码表示报表的系统类型，它与在 NC 平台注册的系统类型编码一致，如下所示为我们的系统类型定义：

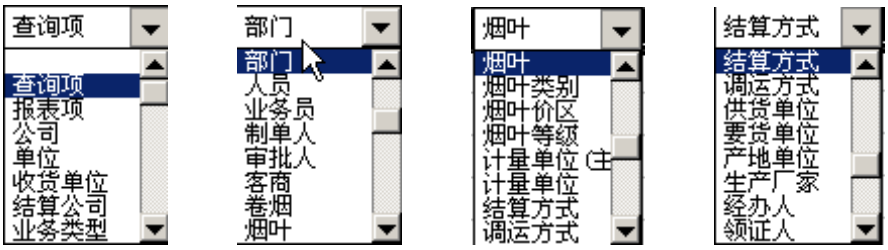
b) 在”报表配置项”页签中填入下面的信息：只需要填写节点编号栏，内容为该报表被分配的节点编号。

B) 配置主表

	A	B	C	D	E	F	G	H	I
1	Name	Code	Data Type	Primary	Foreign Key	Mandatory	是否查询	表名	表别名
2	主键	pk_packcost	char(20)	TRUE	FALSE	TRUE	y	ycca_packcost	a
3	公司主键	pk_corp	char(4)	FALSE	FALSE	FALSE	y		
4	备注	vmemo	varchar(128)	FALSE	FALSE	FALSE	y		
5	部门	pk_deptdoc	char(20)	FALSE	FALSE	FALSE	deptdoc		
6	材料大类	pk_materialcla	char(20)	FALSE	FALSE	FALSE	y		
7	材料档案	vinvmanid	char(20)	FALSE	FALSE	FALSE	y		
8	材料耗量	ninvmannum	decimal(20,8)	FALSE	FALSE	FALSE	y		
9	材料单价	ninvmanprice	decimal(20,8)	FALSE	FALSE	FALSE	y		
10	材料耗用金额	ninvmanmny	decimal(16,4)	FALSE	FALSE	FALSE	y		
11	年度	vyear	char(4)	FALSE	FALSE	FALSE	q		
12	月度	vmonth	smallint	FALSE	FALSE	FALSE	q		
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									

配置 主表 子表1 子表2 报表项 报表SQL 报表组 报表分组SQL 查询模板项 查询模板SQL 报表配置项 报表配置SQL

这个页签的作用是配置主表的原始信息，为构造报表模板的项目做准备。这些内容不需要手工填写，可直接从 PowerDesigner 中拷贝。右上方的表名和表别名起文档的作用，另外工具自动将表别名和项目 Code 结合，组成报表模板的项目名，比如部门在报表模板的名称应为 a_pk_deptdoc。需要特别提醒的是，图中重点标示的一列很重要，它用来定义该字段是否用于查询，所以我们还可以利用它自动生成查询模板。点击是否查询下面的列，会弹出下拉列表，如下图所示：



如果选择报表项，界面会显示”y”，那么该字段作为一个报表项目存在；如果选择查询项，界面会显示”q”，该字段将既作报表项，又做查询项，因此在查询模板中该字段会出现；而下拉列表的其他选项，实际上是表示该字段将做特殊的查询项，特殊之处在于该子段的数据来源于 NC 基本档案的数据表。行业方法将基本档案信息做成了一个比较完备的信息库，比如，在实例中，部门一行的是否查询列显示”deptdoc”，它的意思是要使用部门档案，这样，配置工具会自动带出部门编码和部门名称放到报表模板和查询模板上。后面会着力介绍这个特点。

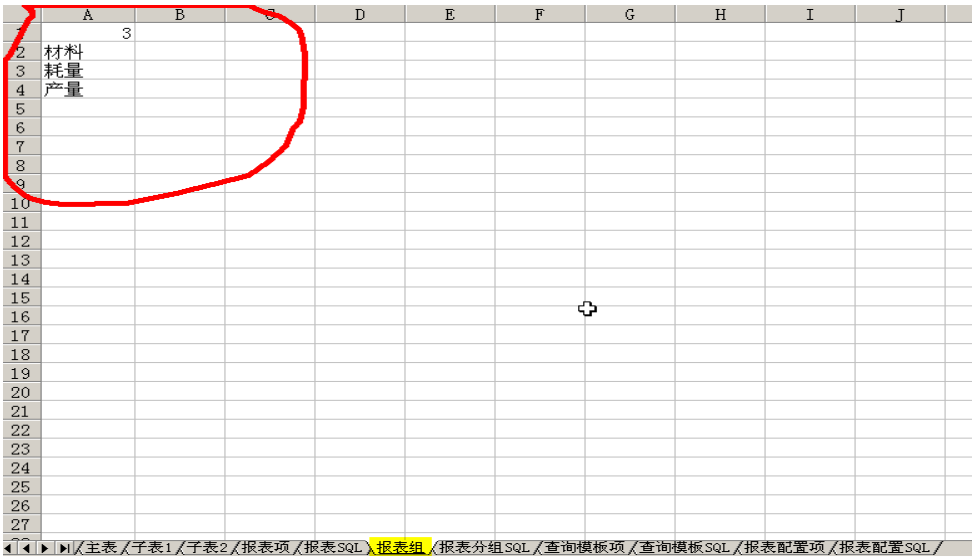
C)　配置子表

	A	B	C	D	E	F	G	H	I
1	Name	Code	Data Type	Primary	Foreign Key	Mandatory	是否查询	表名	表别名
2	主键	pk_packcost_b	char(20)	TRUE	FALSE	TRUE	y	ycca_packcost_b	b
3	公司主键	pk_corp	char(4)	FALSE	FALSE	FALSE	y		
4	分摊主表主键	pk_packcost	char(20)	FALSE	FALSE	FALSE	y		
5	存货ID	pk_invmandoc	char(20)	FALSE	FALSE	FALSE	y		
6	月产量	nmonthoutputnum	decimal(20,8)	FALSE	FALSE	FALSE	y		
7	可分摊产量	nmayoutputnum	decimal(20,8)	FALSE	FALSE	FALSE	y		
8	分摊产量	noutputnum	decimal(20,8)	FALSE	FALSE	FALSE	y		
9	单价	nprice	decimal(20,8)	FALSE	FALSE	FALSE	y		
10	分摊金额	nwastemny	decimal(16,4)	FALSE	FALSE	FALSE	y		
11	分摊耗量	nwastenum	decimal(20,8)	FALSE	FALSE	FALSE	y		
12	月单耗量	nmouthwastenum	decimal(20,8)	FALSE	FALSE	FALSE	y		
13	月单耗金额	nmouthwastemny	decimal(16,4)	FALSE	FALSE	FALSE	y		

这个过程和配置主表一样，我们可以注意到，这里没有对子表定义查询项。

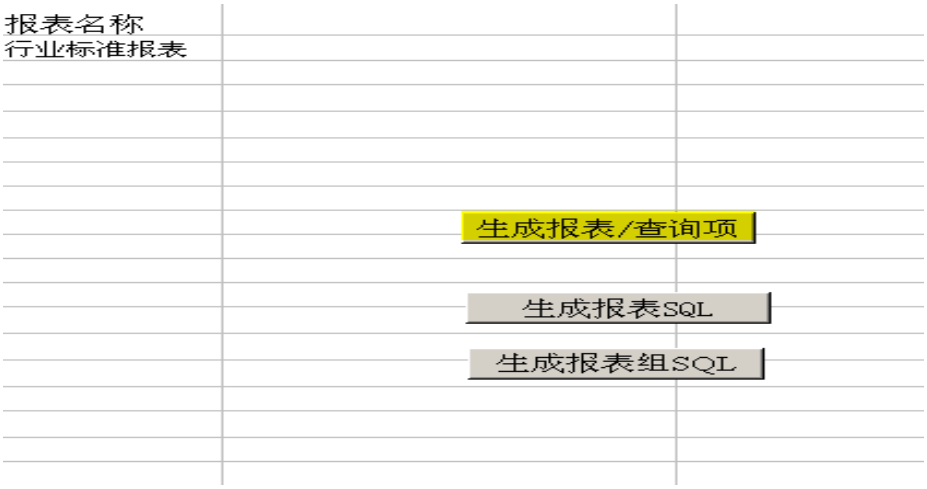
D) 定义报表组

报表组是对报表的一些表头做分组，是显示更为直观。进入”报表组页签”，定义报表组，我们定义三个报表组，依次是材料、耗量、产量，如下所示：



E) 生成报表模板和查询模板

下面我们进入”**报表项**”页签，在该页签中，把页面拉到最右侧，会看到三个按钮，如下所示：



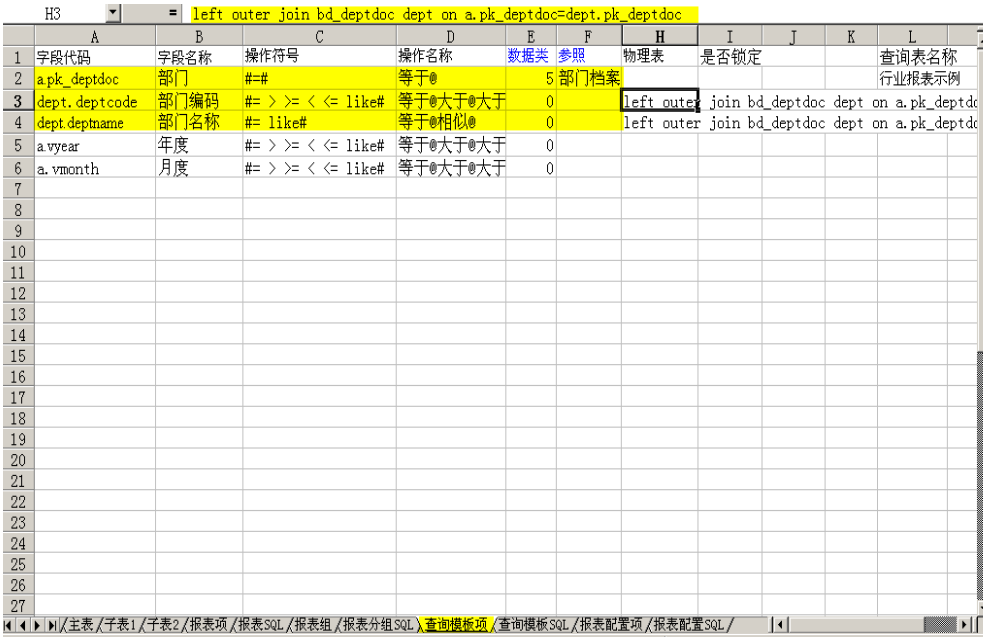
点击”生成报表/查询项”，工具会根据前面的定义生成报表模板和相应的查询模板。

a) 报表模板信息如下：Name、Code、DataType、是否显示、在报表中位置都是自动生成的，显示顺序需要我们根据需要自己设置，在下例中可看到年度、月度会出现在报表的最左端。在公式信息中，如前所述，部门编码和部门名称已被自动带入，这是一个非常强大的功能，因为是公式列，所以在后面的是否原始列信息中，它们的值为 1。后面是与报表组的关联，可以看到，材料耗量、材料单价、材料耗用金额都属于材料组。

	A	B	C	D	E	F	G	H	J	K
1	Name	Code	Data type	是否显示	在报表中位置	显示顺序	公式	是否原始列	报表组	报表名称
2	主键	a_pk_packcost	String	2	1	-1		0		行业标准报表
3	公司主键	a_pk_corp	String	2	1	-1		0		
4	备注	a_vmemo	String	2	1	-1		0		
5	部门	a_pk_deptdoc	String	2	1	-1		0		
6	部门_编码	dept_code	String	1	1	2	dept_code->getCo	1		
7	部门_名称	dept_name	String	1	1	3	dept_name->getCo	1		
8	材料大类	a_pk_materialclass	String	2	1	-1		0		
9	材料档案	a_vinvmanid	String	2	1	-1		0		
10	材料耗量	a_ninvmannum	UFDouble	1	1	4		0	材料	
11	材料单价	a_ninvmanprice	UFDouble	1	1	5		0	材料	
12	材料耗用金额	a_ninvmanmny	UFDouble	1	1	6		0	材料	
13	年度	a_vyear	String	1	1	0		0		
14	月度	a_vmonth	Integer	1	1	1		0		
15	主键	b_pk_packcost_b	String	2	1	-1		0		
16	公司主键	b_pk_corp	String	2	1	-1		0		
17	分摊主表主键	b_pk_packcost	String	2	1	-1		0		
18	存货ID	b_pk_invmandoc	String	2	1	-1		0		
19	月产量	b_nmonthoutputnum	UFDouble	1	1	7		0	产量	
20	可分摊产量	b_nmayoutputnum	UFDouble	1	1	8		0	产量	
21	分摊产量	b_noutputnum	UFDouble	1	1	9		0	产量	
22	单价	b_nprice	UFDouble	1	1	10		0		
23	分摊金额	b_nwastemny	UFDouble	1	1	11		0		
24	分摊耗量	b_nwastenum	UFDouble	1	1	12		0	耗量	
25	月单耗量	b_nmouthwastenum	UFDouble	1	1	13		0	耗量	
26	月单耗金额	b_nmouthwastemny	UFDouble	1	1	14		0		

\配置/主表/子表1/子表2\报表项/报表SQL/报表组/报表分组SQL/查询模板项/查询模板SQL/报表配置项/报表配置SQL/

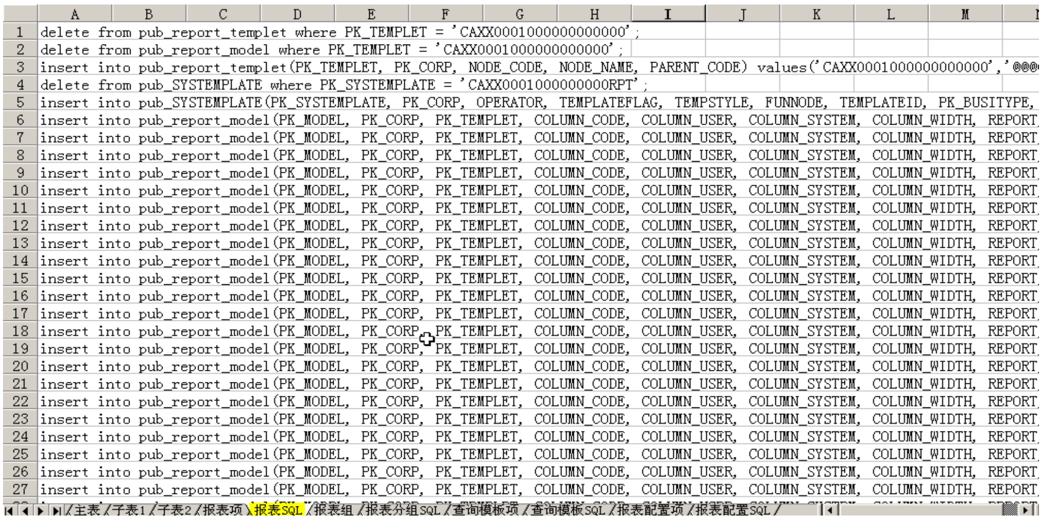
b) 查询模板信息如下：打开”查询模板项”页签，可以看到下面的五个项目将作为查询项出现在查询对话框中。部门编码和部门名称被自动带入，并且自动生成了与之相关的查询语句，这样我们可以不用选择部门参照便可直接输入条件进行查询，大大增强了查询的能力。



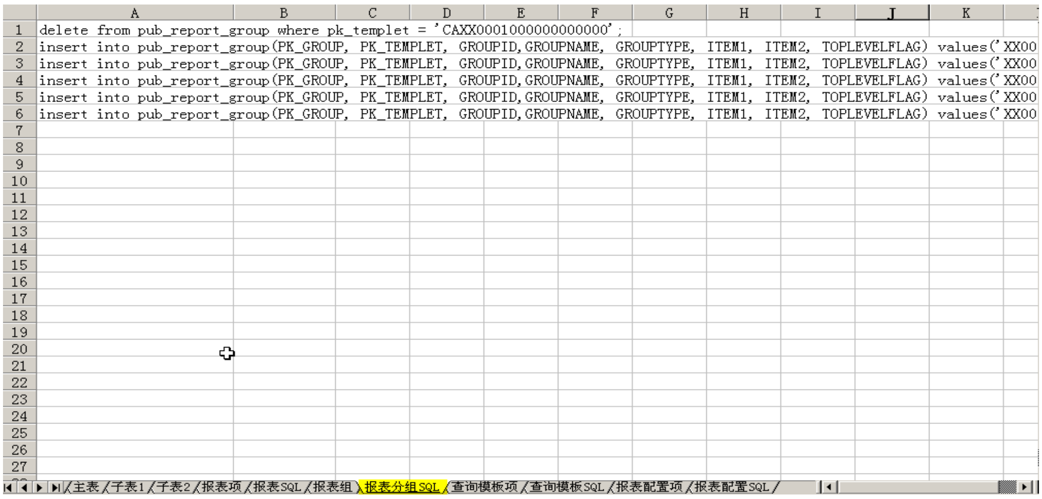
F) 自动生成报表 SQL、报表组 SQL、查询模板 SQL

在前面介绍的”报表项”页签中点击生成报表 SQl、生成报表组 SQL 按钮，在”查询模板项”页签中点击生成查询 SQL 按钮。工具会在”报表 SQL”、”报表组 SQL”、”查询模板 SQL”页签中生成数据插入语句，如下所示，将 SQL 语句拷贝，一次执行击完成配置。

报表模板数据插入 SQL

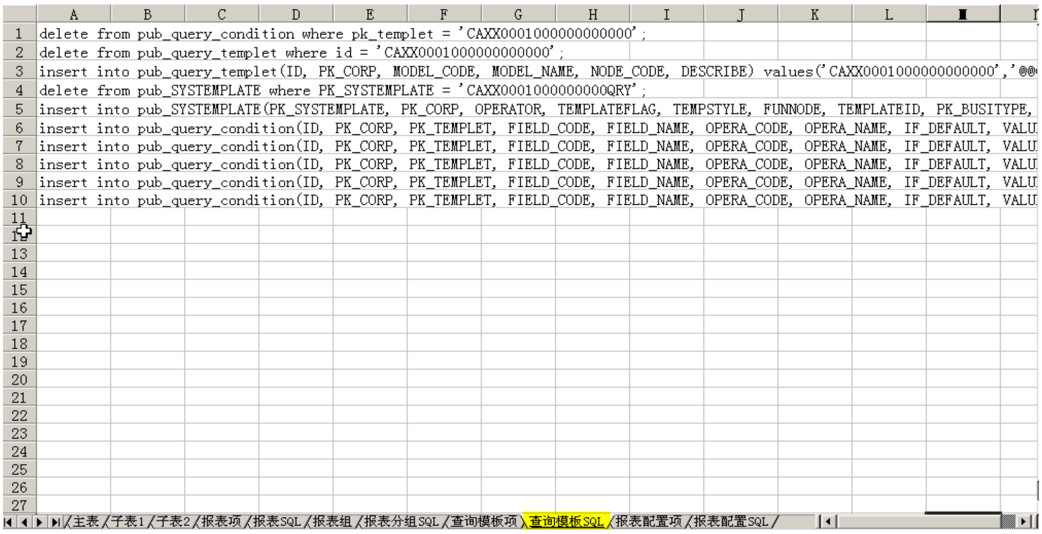


报表组数据插入 SQL



	A
1	delete from pub_report_group where pk_templet = 'CAXX0001000000000000';
2	insert into pub_report_group(PK_GROUP, PK_TEMPLET, GROUPID,GROUPNAME, GROUPTYPE, ITEM1, ITEM2, TOPLEVELFLAG) values('XX00
3	insert into pub_report_group(PK_GROUP, PK_TEMPLET, GROUPID,GROUPNAME, GROUPTYPE, ITEM1, ITEM2, TOPLEVELFLAG) values('XX00
4	insert into pub_report_group(PK_GROUP, PK_TEMPLET, GROUPID,GROUPNAME, GROUPTYPE, ITEM1, ITEM2, TOPLEVELFLAG) values('XX00
5	insert into pub_report_group(PK_GROUP, PK_TEMPLET, GROUPID,GROUPNAME, GROUPTYPE, ITEM1, ITEM2, TOPLEVELFLAG) values('XX00
6	insert into pub_report_group(PK_GROUP, PK_TEMPLET, GROUPID,GROUPNAME, GROUPTYPE, ITEM1, ITEM2, TOPLEVELFLAG) values('XX00

查询模板数据插入 SQL



	A
1	delete from pub_query_condition where pk_templet = 'CAXX0001000000000000';
2	delete from pub_query_templet where id = 'CAXX0001000000000000';
3	insert into pub_query_templet(ID, PK_CORP, MODEL_CODE, MODEL_NAME, NODE_CODE, DESCRIBE) values('CAXX0001000000000000','@@
4	delete from pub_SYSTEMPLATE where PK_SYSTEMPLATE = 'CAXX0001000000000QRY';
5	insert into pub_SYSTEMPLATE(PK_SYSTEMPLATE, PK_CORP, OPERATOR, TEMPLATEFLAG, TEMPSTYLE, FUNNODE, TEMPLATEID, PK_BUSITYPE,
6	insert into pub_query_condition(ID, PK_CORP, PK_TEMPLET, FIELD_CODE, FIELD_NAME, OPERA_CODE, OPERA_NAME, IF_DEFAULT, VALU
7	insert into pub_query_condition(ID, PK_CORP, PK_TEMPLET, FIELD_CODE, FIELD_NAME, OPERA_CODE, OPERA_NAME, IF_DEFAULT, VALU
8	insert into pub_query_condition(ID, PK_CORP, PK_TEMPLET, FIELD_CODE, FIELD_NAME, OPERA_CODE, OPERA_NAME, IF_DEFAULT, VALU
9	insert into pub_query_condition(ID, PK_CORP, PK_TEMPLET, FIELD_CODE, FIELD_NAME, OPERA_CODE, OPERA_NAME, IF_DEFAULT, VALU
10	insert into pub_query_condition(ID, PK_CORP, PK_TEMPLET, FIELD_CODE, FIELD_NAME, OPERA_CODE, OPERA_NAME, IF_DEFAULT, VALU

- 开发步骤

开发步骤分为 3 步：

1) 分析报表数据来源，包括报表需要显示的项目，多表之间的关联。
2) 利用配置工具生成报表模板和查询模板。
3) 利用行业报表 API 创建报表 UI，配置报表剩余信息。

- 代码示例

行业报表 API 提供了两个报表基类，分别处理明细报表和汇总报表，我们分别介绍：

- 明细报表

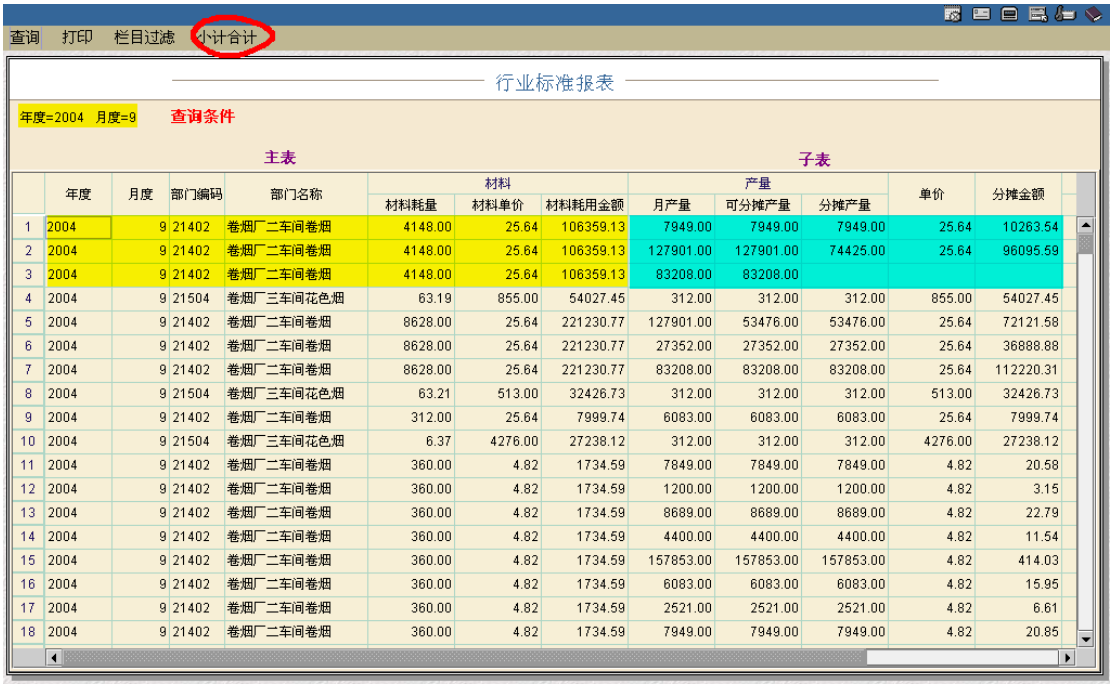
创建报表 UI 类，使其继承 DetailReportBaseEx 类，如下所示：

```
public class HYDetailReportUI extends DetailReportBaseEx{
    protected IReportCtl createIReportCtl() {
        return new IReportCtl() {
```

```
    /**
     * 节点编号
     */
    public String _getModuleCode() {
        return "XX0001";
    }
    /**
     * 获得当前登录的操作人
     */
    public String _getOperator() {
        return null;
    }
    /**
     * 获得当前登录的公司
     */
    public String _getPk_corp() {
        return "1001";
    }
    public String getDefaultSqlWhere() {
        return "";
    }
    public String getDetailReportNode() {
        return null;
    }
    public Class getVOClass() throws Exception {
        return null;
    }
    /**
     * 取得所有表别名
     */
    public String[] getAllTableAlias() {
        return new String[]{
            "a",
            "b"
        };
    }
    /**
     * 取得表连接语句
     */
    public String getTableJoinClause() {
        return "ycca_packcost a left outer join ycca_packcost_b b" +
                " on a.pk_packcost = b.pk_packcost";
    }
};
```

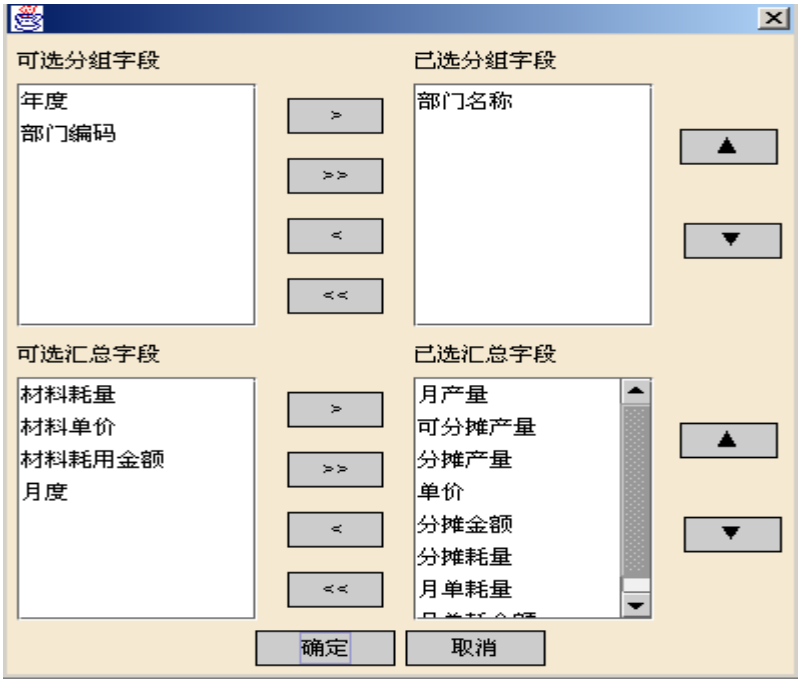
```
    }
}
```

在上面的代码中，我们重点配置了节点编号、表别名、表连接语句等信息，其它的操作都交给基类来处理。其运行结果为：



	年度	月度	部门编码	部门名称	材料耗量	材料单价	材料耗用金额	月产量	可分摊产量	分摊产量	单价	分摊金额
1	2004	9	21402	卷烟厂二车间卷烟	4148.00	25.64	106359.13	7949.00	7949.00	7949.00	25.64	10263.54
2	2004	9	21402	卷烟厂二车间卷烟	4148.00	25.64	106359.13	127901.00	127901.00	74425.00	25.64	96095.59
3	2004	9	21402	卷烟厂二车间卷烟	4148.00	25.64	106359.13	83208.00	83208.00			
4	2004	9	21504	卷烟厂三车间花色烟	63.19	855.00	54027.45	312.00	312.00	312.00	855.00	54027.45
5	2004	9	21402	卷烟厂二车间卷烟	8628.00	25.64	221230.77	127901.00	53476.00	53476.00	25.64	72121.58
6	2004	9	21402	卷烟厂二车间卷烟	8628.00	25.64	221230.77	27352.00	27352.00	27352.00	25.64	36888.88
7	2004	9	21402	卷烟厂二车间卷烟	8628.00	25.64	221230.77	83208.00	83208.00	83208.00	25.64	112220.31
8	2004	9	21504	卷烟厂三车间花色烟	63.21	513.00	32426.73	312.00	312.00	312.00	513.00	32426.73
9	2004	9	21402	卷烟厂二车间卷烟	312.00	25.64	7999.74	6083.00	6083.00	6083.00	25.64	7999.74
10	2004	9	21504	卷烟厂三车间花色烟	6.37	4276.00	27238.12	312.00	312.00	312.00	4276.00	27238.12
11	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	7849.00	7849.00	7849.00	4.82	20.58
12	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	1200.00	1200.00	1200.00	4.82	3.15
13	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	8689.00	8689.00	8689.00	4.82	22.79
14	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	4400.00	4400.00	4400.00	4.82	11.54
15	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	157853.00	157853.00	157853.00	4.82	414.03
16	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	6083.00	6083.00	6083.00	4.82	15.95
17	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	2521.00	2521.00	2521.00	4.82	6.61
18	2004	9	21402	卷烟厂二车间卷烟	360.00	4.82	1734.59	7949.00	7949.00	7949.00	4.82	20.85

行业明细报表基类提供了小计合计功能，如下所示：



我们选择按”部门名称”分组， 并选择了多个子表字段做汇总，其运行结果为：

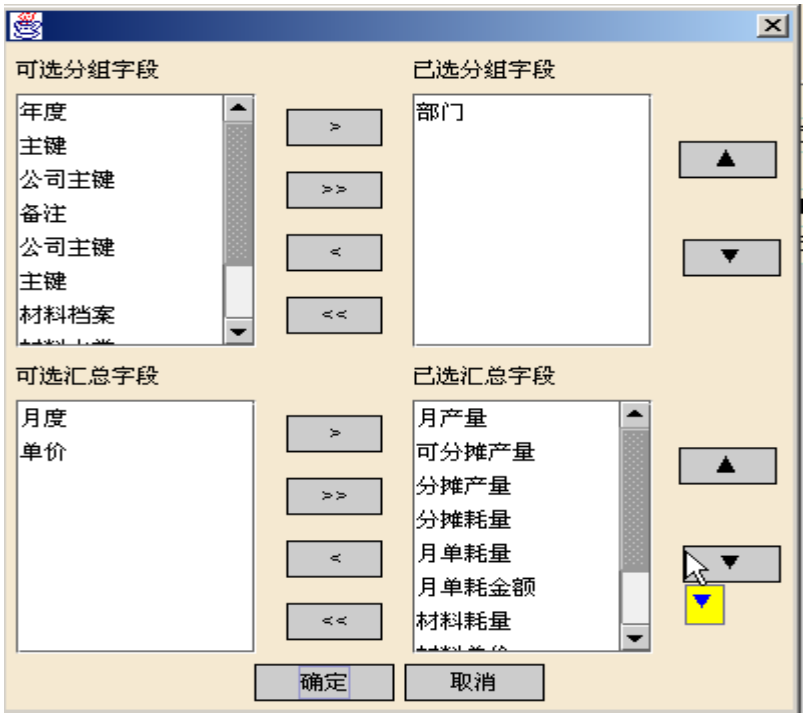
行业标准报表

年度=2004 月度=9

	年度	月度	部门编码	部门名称	材料			产量			单价	分摊金额
					材料耗量	材料单价	材料耗用金额	月产量	可分摊产量	分摊产量		
34	2004	9	21503	卷烟厂三车间卷烟	766.32	30.78	23585.82	3833.00	3833.00	3833.00	30.78	15120.33
35	2004	9	21503	卷烟厂三车间卷烟	1619.20	18.74	30336.49	8814.00	8814.00	8814.00	18.74	30336.49
36	2004	9	21503	卷烟厂三车间卷烟	208.54	3333.33	695133.33	3833.00	3833.00	3412.00	3333.33	695133.33
37	2004	9	21503	卷烟厂三车间卷烟	25.40	3333.33	84666.67	3833.00	421.00	421.00	3333.33	84666.67
38	2004	9	21503	卷烟厂三车间卷烟	9.50	4545.75	43184.64	159.00	159.00	159.00	4545.75	43184.64
39	2004	9	21503	卷烟厂三车间卷烟	99.69	1667.00	166174.89	1987.00	1987.00	1987.00	1667.00	166174.89
40	2004	9	21503	卷烟厂三车间卷烟	4667.00	13.00	60671.00	3833.00	3833.00	3833.00	13.00	60671.00
41	2004	9	21503	卷烟厂三车间卷烟	2068.00	10.00	20680.00	1987.00	1987.00	1987.00	10.00	20680.00
42	2004	9	21503	卷烟厂三车间卷烟	140.00	7.67	1073.33	159.00	159.00	134.00	7.67	1073.33
43	2004	9	21503	卷烟厂三车间卷烟	25.00	11.00	275.00	159.00	25.00	25.00	11.00	275.00
44	2004	9	21503	卷烟厂三车间卷烟	8857.00	7.03	62226.10	8814.00	8814.00	8814.00	7.03	62226.10
45				小计				159588.00	152630.00	151763.00	69566.25	2286320.65
46	2004	9	21504	卷烟厂三车间花色烟	63.19	855.00	54027.45	312.00	312.00	312.00	855.00	54027.45
47	2004	9	21504	卷烟厂三车间花色烟	63.21	513.00	32426.73	312.00	312.00	312.00	513.00	32426.73
48	2004	9	21504	卷烟厂三车间花色烟	6.37	4276.00	27238.12	312.00	312.00	312.00	4276.00	27238.12
49	2004	9	21504	卷烟厂三车间花色烟	80.89	1110.00	89787.90	399.00	399.00	399.00	1110.00	89787.90
50	2004	9	21504	卷烟厂三车间花色烟	80.46	513.00	41275.98	399.00	399.00	399.00	513.00	41275.98
51	2004	9	21504	卷烟厂三车间花色烟	8.04	5128.00	41229.12	399.00	399.00	399.00	5128.00	41229.12

- **汇总报表**

创建报表 UI 类，使其继承 TotalReportBaseEx 类，其他代码与明细报表一样。在汇总报表中，做查询前应先进行汇总设置，如下所示：



我们选择按部门分组，并对主表和字表的数据列进行汇总，其运行结果为：

	部门编码	部门名称	月产量	可分摊产量	材料			产量			材料耗用金额	分摊
					分摊产量	分摊耗量	月单耗量	月单耗金额	材料耗量	材料单价		
1	21402	卷烟厂二车间卷烟	8694862.00	7135384.00	6006528.00	1413686.82	46.10	3747.18	5938371.25	305823.25	18412358...	902899
2	21503	卷烟厂三车间卷烟	159588.00	152630.00	151763.00	34458.07	9.77	928.42	42150.18	69566.25	2473712.76	22863
3	21504	卷烟厂三车间花色烟	50594.20	49934.42	49274.20	15086.94	31.94	3917.57	23013.17	71994.59	2503937.57	23134

- **总结**

这章介绍了行业报表开发方法，它既包括了对一类报表公共特性的抽取，又包括了一套强大的配置工具，在制作前面应用场景中介绍的报表时，具有非常明显的优势。

1.2 分块填充报表

- **简述**

此类报表从逻辑上可在纵向和横向划分为若干分块，同属一个纵向分块的数据对应同一个分类，同属一个横向分块的数据对应同一个业务系统的信息，但并不要求位于同一行的记录存在精确的匹配关系，因为用户只关心每个大行（纵向分块）包含哪些信息。

- **应用场景**

	X_{A1}	…	X_{An}	X_{B1}	…	X_{Bn}	……	X_{Z1}	…	X_{Zn}
Y_{A1}										
…										
Y_{Am}										
Y_{B1}										
…										
Y_{Bm}										
……										
Y_{Z1}										
…										
Y_{Zm}										

在上述的分块报表中，纵向的每一块代表一种分类及其相关的信息，在同一纵向块中的各横向块具有某个（或某几个）共同的键值，但并不要求每行的各横向块记录存在对应关系，即对这些记录的上下顺序并不明感。比如下面的表样：

存货信息			制成品信息			库存信息		
编码	名称	计量	品名	单价	型号	日期	入库数	出库数
01	铜版纸	张	画册	15.00	A3	10-15	200	
			海报	30.00	A2			
02	塑料	千克	塑料袋	1.00	P33	10-15	10	
			塑料布	2.00	Q33	10-16		20
						10-17		30

其中，一种存货可能对应多种制成品，同时也可能对应多条库存记录，这些相关信息显示在一个纵向分块中（比如黄色部分）。而处在同一行的数据未必存在对应关系，比如塑料袋和10-15 的库存信息没有联系，它们只是都与同一种存货（塑料）相关。

- **适用性和不适用性**

适用于：

- 涉及的业务系统存在外键关系的综合明细报表；

不适用于：

- 各列数据必须满足精确行 ID 才能成为一行的数据集合；

- **解决方案**

采用行业报表提供的 VOFILL 方案。开发者根据实际业务逻辑，继承指定基类编写若干子类代码，描述分块和数据填充的规则，将待填充的数据（多个 CircularlyAccessibleValueObject）哈希化，由基类进行填充，然后再反哈希化成为一个 CircularlyAccessibleValueObject，这就是报表最终展现的数据集。如果数据有特殊处理要求，可以在哈希化和反哈希化过程中进行修补。

- **开发步骤**

这里我们只着重阐述如何把多个 CircularlyAccessibleValueObject 按照开发者描述的填充规则合成一个 CircularlyAccessibleValueObject 的开发过程，而不去关注开发者如何获得那多个 CAVO 以及如何展现最终填充出来的 CAVO。之所以这样做，是因为开发者有多种途径去获得和展现 CAVO，比如报表模板、查询引擎或自行开发等等，如果一一赘述会偏离本模式的主旨。另外，CAVO 是 NC 广泛应用的一个基础数据结构，这里也不再对其进行详细说明，我们在下文中使用的数据结构主体是 nc.vo.trade.report.ReportVO，它是 nc.vo.pub.CircularlyAccessibleValueObject 的非抽象子类。

第一步：获得各横向分块对应的 CAVO（略）；

第二步：编写 VO 填充类。每个横向分块对应一个 VO 填充类，继承自 nc.vo.trade.report.vofill.AbstractVOFill，在重载的方法 getTargetAndSourceFields 中描述源字段到填充列的对照关系；

第三步：编写报表填充工厂类。继承自 nc.vo.trade.report.vofill.AbstractReportFactory，在重载的方法 createIVOFill 中描述分块规则，createReportVOMetaClass 中描述列结构信息；

第四步：哈希化上一步获得的各个 CAVO（CircularlyAccessibleValueObject[][]→HashMap[]）。这里要指定各个横向分块的联系纽带（外键），详见代码示例；

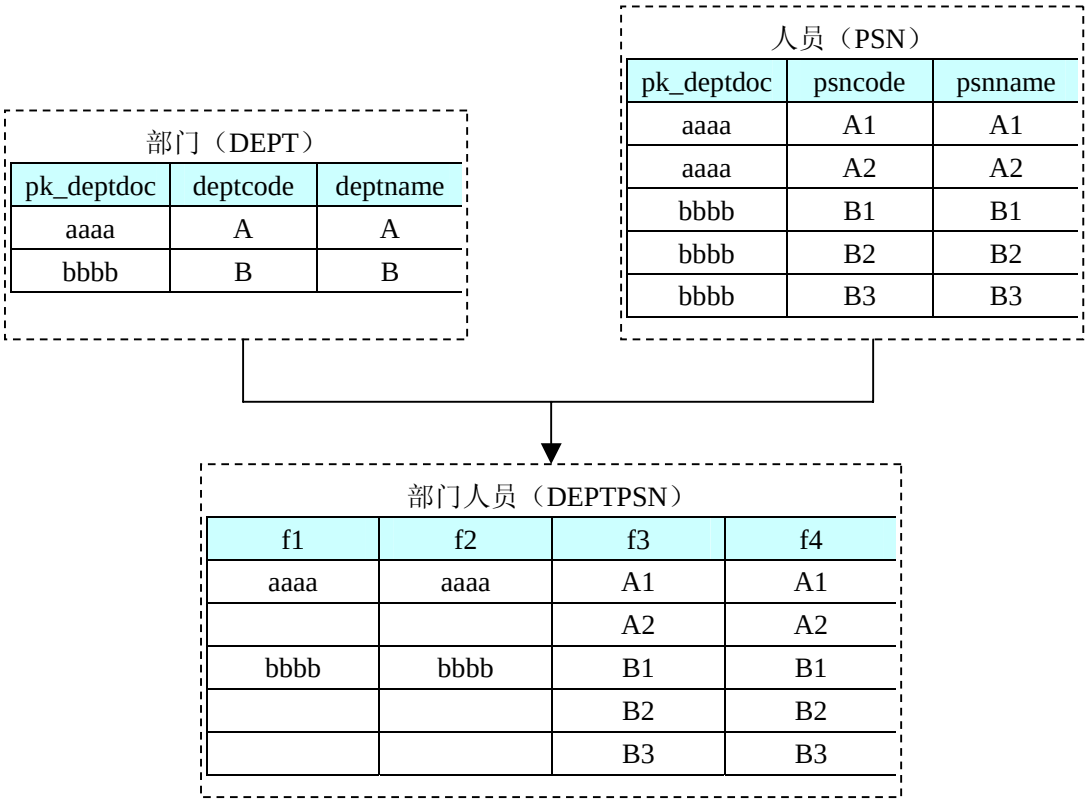
第五步：执行数据填充（HashMap[]→HashMap）。缺省类 nc.vo.trade.report.vofill.DefaultBillFill 可以解决绝大部分问题；

第六步：反哈希化（HashMap→CircularlyAccessibleValueObject[]）；

第七步：展现填充好的 CAVO（略）；

- **开发范例**

部门与人员明细报表



1、 编写一个常数类

```
public class Consts {
    public final static String DEPTID = "pk_deptdoc";
    public final static String DEPTCODE = "deptcode";
    public final static String DEPTNAME = "deptname";
    public final static String OWNERDEPTID = "pk_deptdoc";
    public final static String PSNCODE = "psncode";
```

```
        public final static String PSNNAME = "psnname";
        public final static String F1 = "f1";
        public final static String F2 = "f2";
        public final static String F3 = "f3";
        public final static String F4 = "f4";
    }
```

2、 编写部门 VO 填充类：

```
    public class DeptVOFill extends AbstractVOFill
```

重载源列与目标列对照方法：

```
    protected String[][] getTargetAndSourceFields() {
        return new String[][] {
            new String[] { Consts.F1, Consts.DEPTCODE },
            new String[] { Consts.F2, Consts.DEPTNAME }
        };
    }
```

3、 编写人员 VO 填充类：

```
    public class PsnVOFill extends AbstractVOFill
```

重载源列与目标列对照方法：

```
    protected String[][] getTargetAndSourceFields() {
        return new String[][] {
            new String[] { Consts.F3, Consts.PSNCODE },
            new String[] { Consts.F4, Consts.PSNNAME }
        };
     }
```

4、 编写报表填充工厂类：

```
    public class DeptPsnRptFactory extends AbstractReportFactory
```

增加两个常量：

```
    public final static int DEPT = 0;
    public final static int PSN = 1;
```

重载分块规则描述方法：

```
    public IVOFill createIVOFill(int sourceDataType) {
        switch (sourceDataType) {
            case DEPT :
                return DeptVOFill.getInstance();
            case PSN :
                return PsnVOFill.getInstance();
        }
```

```
        throw new IllegalArgumentException("不支持的数据填充类型");
    }
```

重载列结构描述方法：

```
    public ReportVOMetaClass createReportVOMetaClass() {
        String[] fields = new String[] { Consts.F1, Consts.F2, Consts.F3, Consts.F4 };
        int[] datatypes =
            new int[] {
                IUFtypes.STRING, IUFtypes.STRING,
                IUFtypes.STRING, IUFtypes.STRING };
        ReportVOMetaClass voClass =
            new ReportVOMetaClass(fields, fields, datatypes, new String[] { "" }, "");
        return voClass;
    }
```

5、 编写哈希化方法，注意横向分块纽带字段的设置：

（import nc.vo.trade.summarize.*;）

```
    protected static ArrayList hashlizeVOs(
        CircularlyAccessibleValueObject[] cas1,
        CircularlyAccessibleValueObject[] cas2) {

        ArrayList al = new ArrayList();
        HashMap map = null;
        //部门
        if (cas1 != null && cas1.length != 0) {
            map =
                Hashlize.hashlizeObjects(
                    cas1,
                    new VOHashKeyAdapter(new String[] { Consts.DEPTID }));
        }
        al.add(map);
        //人员
        if (cas2 != null && cas2.length != 0) {
            map =
                Hashlize.hashlizeObjects(
                    cas2,
                    new VOHashKeyAdapter(new String[] { Consts.OWNERDEPTID }));
        }
        al.add(map);
        //
        return al;
    }
```

6、 编写 VO 填充方法：

（import nc.vo.trade.report.vofill.*;）

```
protected static HashMap fillVOs(ArrayList al) {

    HashMap resultHash = new HashMap();
    int[] sourceDataTypes = { DeptPsnRptFactory.DEPT, DeptPsnRptFactory.PSN };
    //循环填充
    DefaultBillFill bFill = new DefaultBillFill(DeptPsnRptFactory.getInstance());
    for (int i = 0; i < al.size(); i++)
        if (al.get(i) != null)
            bFill.fillBill(resultHash, (HashMap) al.get(i), sourceDataTypes[i]);
    return resultHash;
}
```

7、 编写反哈希化方法：

```
protected static ReportVO[] extractVOs(HashMap billHash) {

    ArrayList keys_al = new ArrayList();
    keys_al.addAll(billHash.keySet());
    //排序
    //Collections.sort(keys_al);

    ArrayList al = new ArrayList();
    for (int i = 0; i < keys_al.size(); i++) {
        Object key = keys_al.get(i);
        ArrayList tempAl = (ArrayList) billHash.get(key);
        al.addAll(tempAl);
    }
    return (ReportVO[]) al.toArray(new ReportVO[0]);
}
```

8、 编写一个拼装 CAVO 的方法，调用上述函数：

```
public static ReportVO[] getData(
    CircularlyAccessibleValueObject[] cas1,
    CircularlyAccessibleValueObject[] cas2)
    throws Exception {

    /** 1、哈希化 */
    ArrayList al = hashlizeVOs(cas1, cas2);

    /** 2、VO 填充 */
```

```
        HashMap resultHash = fillVOs(al);

        /** 3、反哈希化 */
        ReportVO[] reportVOs = extractVOs(resultHash);
        return reportVOs;
    }
```

9、 获得部门和人员信息的 CAVO，利用上述方法做分块填充，然后展现：

部门档案主键	部门编码	部门名称
1001AA1000000000000RU	2111	开发部01
1001AA1000000000000RW	2102	开发部02
1001AA1000000000000RX	2201	服务支持部01
1001AA1000000000006KD	0501	HR产品开发部
1001AA1000000000006KP	0502	HR产品市场部
1001AA1000000000006KQ	0503	HR服务支持部
1001BA0000000000FQF	21	开发部
1001BA0000000000FQG	22	服务支持部
1002AA1000000000034C	30	大堂
1002AA1000000000034N	40	饭店餐厅
9999AA0000000000001	01	直属职能部门
9999AA0000000000002	0101	总裁办
9999AA0000000000004	0103	人力资源部
9999AA0000000000007	010801	行政下属财务部
9999AA0000000000009	0108	行政部
9999AA0000000000050	03	ERP事业部
9999AA0000000000051	0301	ERP产品管理部
9999AA0000000000052	0302	ERP产品市场部

部门主键	人员编码	姓名
9999AA0000000000007	7	闫红
9999AA0000000000078	15	杨冰
9999AA0000000000002	000011	吴平
9999AA0000000000009	22	张宇
9999AA0000000000078	24	洪山
9999AA0000000000082	25	毛杰
9999AA0000000000082	27	许文
9999AA0000000000009	31	汤伟
9999AA0000000000082	32	张炯
9999AA0000000000007	40	王平
9999AA0000000000082	42	李德
9999AA0000000000073	48	隋华
9999AA0000000000079	53	汪杰
9999AA0000000000079	55	李陶
9999AA0000000000079	57	敖平
9999AA0000000000079	62	邵晴
9999AA0000000000004	83	周争
9999AA0000000000007	182	余捷
9999AA0000000000009	186	曹保

	f1	f2	f3	f4
1	0307	ERP生产制造产品部	295	谢华
2			375	张军
3			721	司红
4			739	苑科
5			804	宋果
6			857	张友
7			869	张杰
8			885	李忠
9			890	李俊
10			1081	王占
11	0305	ERP供应链产品部	417	谢阳
12			432	王军
13			456	江霞
14			485	赵宇
15			560	李巧

2. 基于 CELL REPORT 工具开发的报表

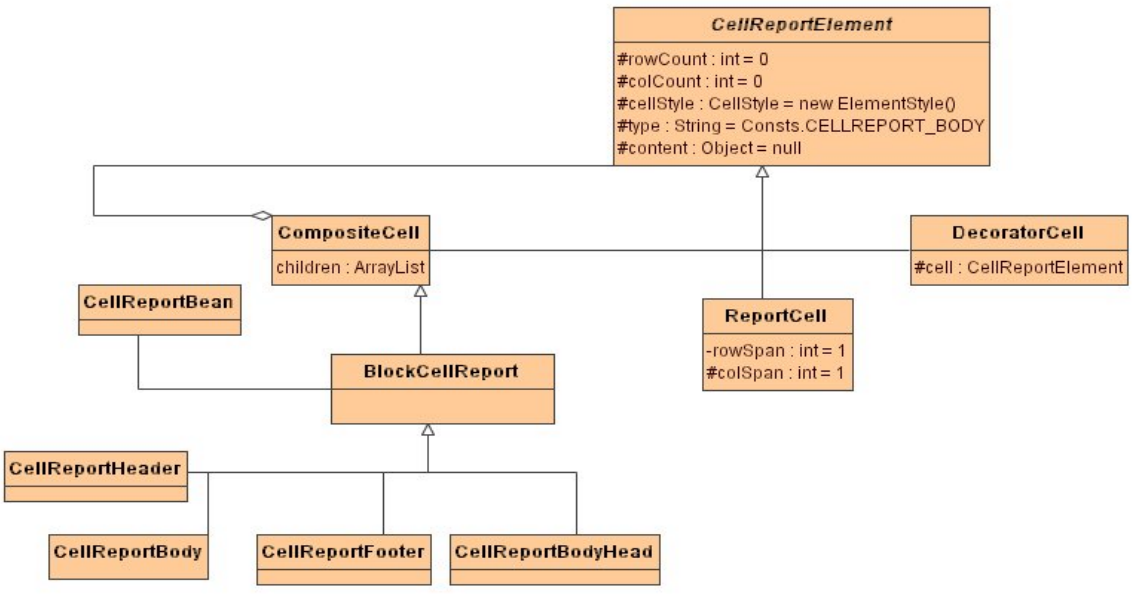
2.1 CELL REPORT 工具介绍

在报表开发中，有可能会遇到一些比较怪异的报表，它们不符合常规的数据展示模型，或者超越了报表模版的功能范围，因此给开发者带来了很多难以解决的问题。如果从更深刻的角度来观察这种报表，可以得出的结论是它们非常具有个性化，从而失去了与大众标准产生关联的基础。这种个性化，主要体现在数据模型和展示模型上：

1) 数据模型的个性化：最富有代表性的场景是报表的数据来自多个离散的数据源，也就是说，报表的每个单元格都有不同的取数规则。
2) 展示模型的个性化：客户在报表里希望看到最直观的东西和最重要的东西，要满足直观性就必须定制，并且具备将数据模型中的数据按任意方式展现出来的能力。这实际上对报表的展现功能提出了很高的要求。

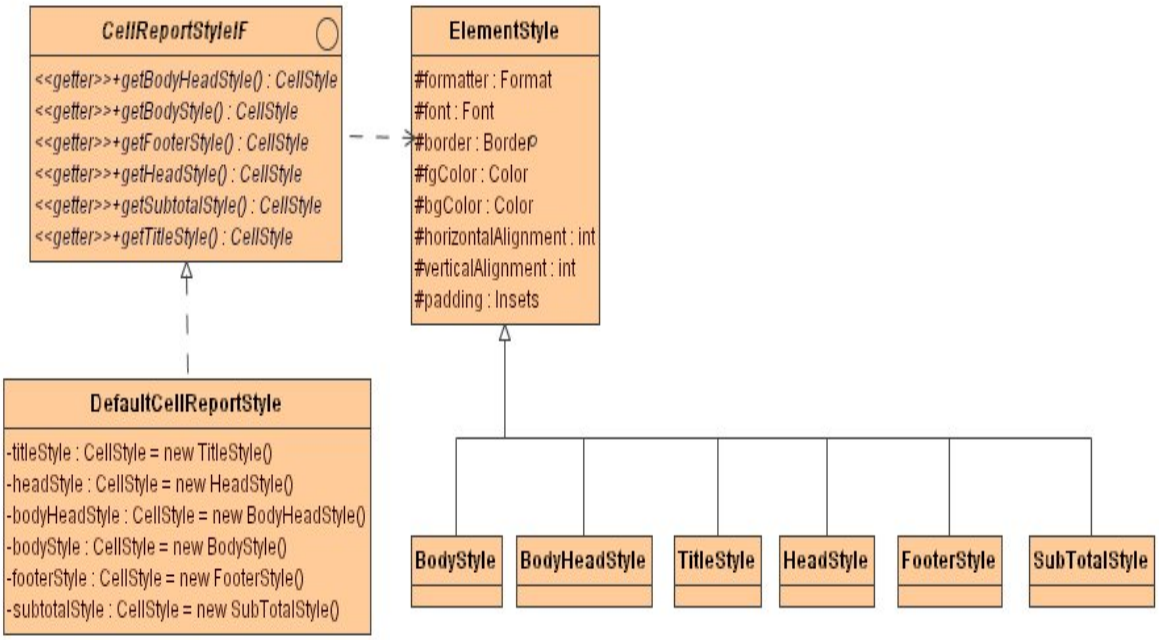
针对报表开发存在的这两种个性化需求，我们采用了一种新的报表 GUI——Cell Report，它将开源项目 JGrid 的 API 进行了封装，并且补充了很多适合于报表开发的功能。Cell Report 的特点是可以对报表的每个单元格进行个性化渲染，并且可以将多个相邻单元格组成“块”，这样，我们可以将一行的数据或一列的数据作为一个“块”统一管理，为带有动态行、动态列，以及在数据展示上有特殊要求的报表提供了解决方案。另外，Cell Report 结合了开源项目 JEP，为报表增加了强大的公式功能，这个功能可以为每个单元格提供取数公式，从而满足了数据模型的个性化。

- **Cell Report API 介绍及相关工具**
  - Cell Report 类图
    A) 基础结构类图



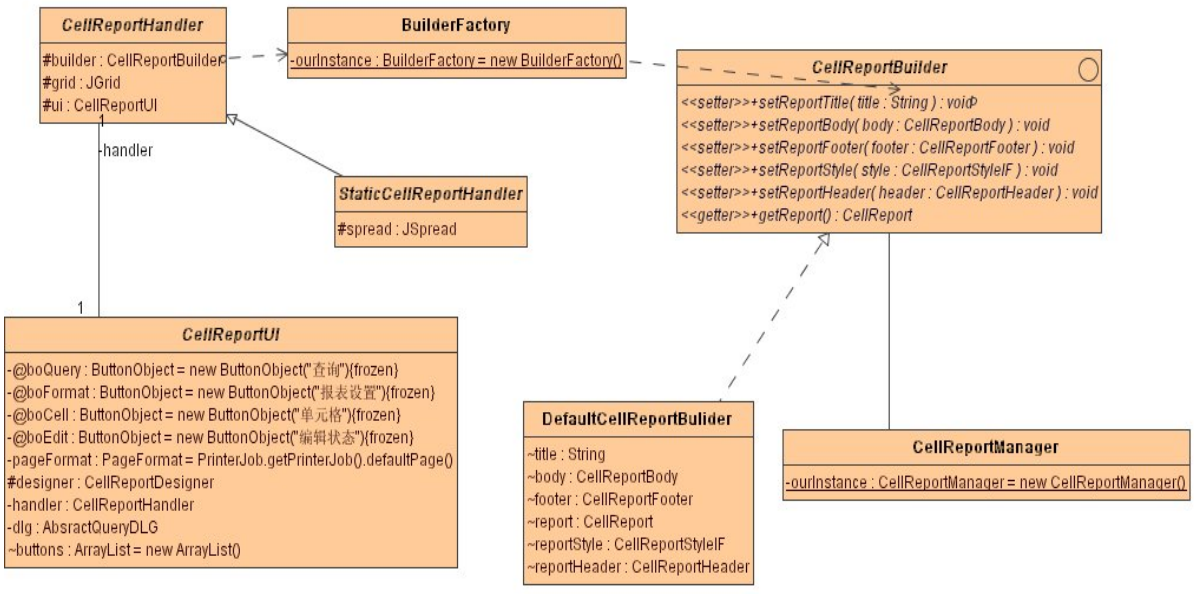
上图可以看到，每个单元格类 ReportCell 都继承自抽象基类 CellReportElement，同时每一个报表块类 BlockCellReport 利用 Composite 模式复合进多个嵌套的报表块。另外，根据报表不同的展示逻辑，分别创建了报表头、报表尾、报表体等报表组成元素，它们实际上都是报表块，所以都继承自 BlocjCellReport 类。

B) 样式库类图



每个单元格都可以设置显示的属性，比如字体、前景、背景、对齐方式等，这些信息被抽象成 ElementStyle 类，同时，我们可能需要对报表逻辑块整体实施样式，因此创建了报表头样式类、报表体样式类等等，它们可以很好地被重用。

C) 面向程序员的开发接口



CellReportUI 类是报表开发的 UI 基类， 它负责报表界面展示——比如按钮、设置工具等等。CellReportHandler 类负责组织数据并构造报表的各个部分。

➢ Cell Report 报表设计器

这些按钮可对报表展示界面进行调整，并且可将当前报表的显示存储成报表设计文件。

2.2 CELL REPORT 报表

- **简述**

参见上一小节。

- **应用场景**

就目前的实际开发经验告诉我们，绝大部分的报表都可以用 NC 报表模板完成，除非下面两种情况：

1) **报表数据的来源呈离散状态，不能通过简单的多表关联取得所有数据。**比如：

项目	月度金额	季度金额	年度金额	全部金额
收入	A_MONTH	A_QUARTER	A_YEAR	A_TOTAL
支出	B_ MONTH	B_ QUARTER	B_YEAR	B_TOTAL
上缴	C_ MONTH	C_ QUARTER	C_YEAR	C_TOTAL

我们分析一下上表的数据来源：该报表要展示三个项目的月度、季度、年度和全部金额。这三个项目的数据分别存储在三个数据表中，而月度、季度、年度、全部肯定对应着四个不同的查询条件——月度查询当前月，季度需要将当前季度的三个月做小计。因此，这个报表的数据是离散的，我们很难用一个查询语句获得所有数据。解决的办法是为每个单元格设置一个有意义的公式，公式中包含了查询条件、查询的数据表等信息，当要绘制单元格时，运行公式取得正确数据。

2) **报表数据的展示非常特殊，很难用单据模板完成。**报表的样式决定于客户想看到什么，有些时候，客户希望把他认为最重要的东西用一个特殊的方法展示出来，这种需求往往超越了我们对报表格式的定义。基于报表模板的开发对此是无能为力的，因为这些特殊需求只能用定制来满足。要做到灵活的界面的定制，意味着我们能灵活地控制报表单元格、报表行和报表列。因此，如果遇到这类需求，应该使用本章介绍的方法制作报表。

这两种情况只是从大的方面做出的区分。在实际报表开发中，应首先对报表需求做详尽的分析，再慎重做出选择。比如，有一些报表从表面看数据组织得非常复杂，但如果对其数据来源进行深入分析的话，其实可以利用报表模板的数据交叉方法快速地开发出来，此时就没有必要采用基于 Cell Report 的开发方法。因此，遇到方案的选择问题时，最为重要的是慎重地根据应用场景划分界限。在这里，我们重申本章介绍的报表开发方法的适用性和不适用性。

- **适用性和不适用性**

适用于：

- 报表数据模型的数据来源非常离散。
- 报表展示模型具有很强的个性化，需要定制。

不适用于：

- 它可以开发出所有的报表，但与报表模板相比，它是一种重量级的方法，因为我们需要通过写代码来完成，而不能通过一套配置工具。因此，从开发效率角度而言，它不适用于可以通过报表模板配置出的报表。
- 对打印有严格要求的报表。目前这套基于 Cell Report 的报表对打印支持很弱，它采用所见即所得的方式。另外，它不能和打印模板集成，在页面分页时不提供动态表头支持，批量打印需要自己写代码实现。

- **解决方案**

在前面应用场景中我们分析了两种场景，分别针对数据模型的离散化和展示模型的个性化，那么，在这里我们也提供两种相应的方案来分别解释如何利用 Cell Report 实现报表开发。

1) 报表展示模型个性化的解决方案：Cell Report 本身是一套面向程序员的报表 GUI 开发工具，我们利用它提供的 API 编写代码，构造报表头、报表体、报表尾。另外，Cell Report 利用了分“块”的概念，我们可以将多个单元格作为一个整体操作，并可以灵活地定义它们的样式风格。
2) 报表数据模型离散化的解决方案：Cell Report 集成了 JEP 的公式功能，我们可以定义每个单元格的取数公式（公式的定义也非常关键），然后将整张报表存成*.report 的配置文件格式（二进制），在进行单元格渲染时，会自动运行公式并将取得的数据展现。

- **开发步骤**

1) 展示模型个性化报表的开发步骤：
   A) 分析报表大体由哪几个“块”组成，哪一部分是报表头，哪一部分是报表体，哪一部分是报表尾等等。
   B) 报表体通常比较复杂，分析报表体的行和列有什么特点。
   C) 编码构造报表头和报表尾。
   D) 根据报表体的行列特点，编码构造报表体。
2) 数据模型离散化报表的开发步骤：
   A) 分析数据来源，定义好报表取数公式。
   B) 利用 Cell Report 的报表设计工具，定义好报表的展示模型，并在相应单元格中添入先前定义的取数公式。
   C) 存储设计好的报表为*.report 文件，它将包含展示模型的信息和公式信息。
   D) 如有必要，编写代码，控制报表数据。（比如，利用查询对话框条件查询数据）

- 开发范例

  - **展示模型个性化报表的开发示例**

    1. 创建 UI 类，继承 CellReportUI 类：UI 类非常简单，用它取得查询对话框和报表处理类即可，如下：

```
public class DynamicShowReportUI extends CellReportUI{
    /**
     * 创建报表处理器类
     */
    protected CellReportHandler createCellReportHanlder()
            throws CellReportException
    {
        return new DynamicShowReportHandler(this);
    }

    /**
     * 创建报表查询对话框
     */
    protected AbsractQueryDLG createQueryDLG() {
        return new DynamicShowQueryDLG();
    }

    /**
     * 创建报表标题
     */
    public String getTitle() {
        return "展示模型个性化报表";
    }
}
```

    2. 创建查询对话框类，继承 AbstractQueryDLG 类

```
public class DynamicShowQueryDLG extends AbsractQueryDLG{

    /** 日期参照 **/
    UIRefPane dateRef = new UIRefPane();
    private String year = null;
    private String month = null;

    public DynamicShowQueryDLG() {
        // 初始化对话框组件
        init();
    }

    /**
```

```
     * 在界面上放置组件
     */
    protected Map getComponentMap() {
        dateRef.setRefNodeName("日历");
        Map map = new HashMap();
        map.put("日期", dateRef);
        return map;
    }

    /**
     * 将查询相关信息放入报表应用程序共享的上下文中
     */
    protected void updateCondition() {
        year = String.valueOf(new UFDate(dateRef.getText()).getYear());
        month = String.valueOf(new UFDate(dateRef.getText()).getMonth());
        addReportCondition("YEAR", year);
        addReportCondition("MONTH", month);
    }

    /**
     * 检查是否添入了合适的查询条件
     */
    public boolean checkCondition() {
        String date = dateRef.getText();
        if (date == null || date.length() == 0)
            return false;
        else
            return true;
    }
}
```

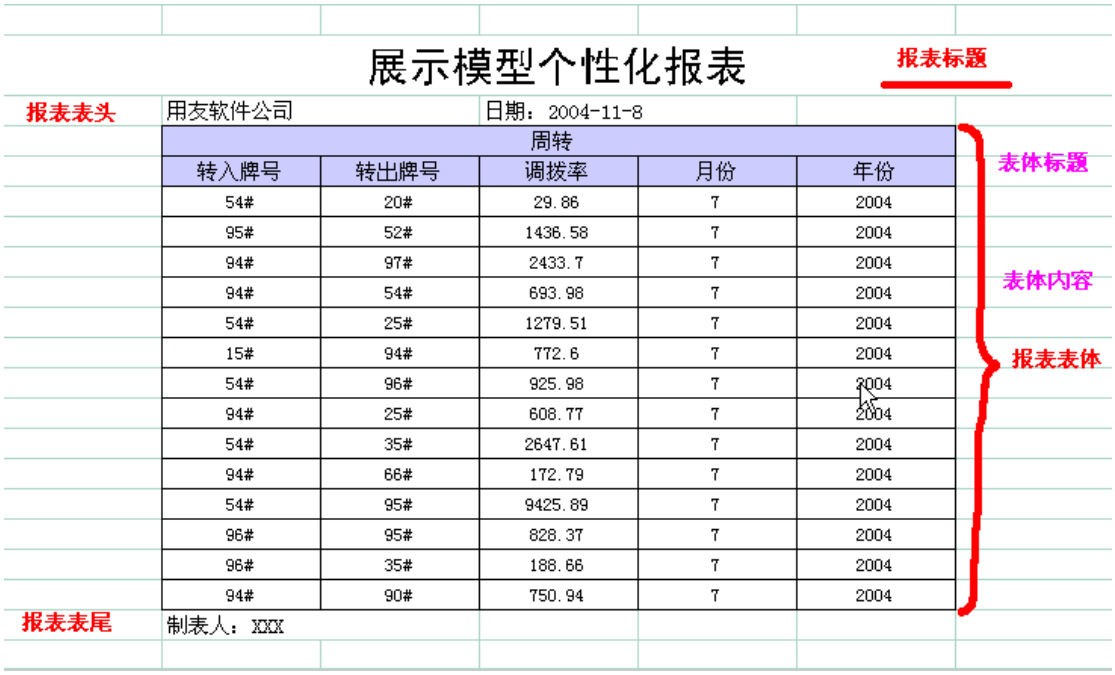
3. 创建报表处理类，继承 CellReportHandler 类

```
public class DynamicShowReportHandler extends CellReportHandler{

    public DynamicShowReportHandler(CellReportUI ui) {
        super(ui);
    }

    /**
     * 设置报表标题
     */
    protected String createReportTitle() throws CellReportException {
        return "展示模型个性化报表";
    }
```

```
/**
 * 创建报表头
 */
protected CellReportHeader createReportHeader() throws CellReportException {
      CellReportHeader header = new CellReportHeader(4);
      header.setCellValueAt("用友软件公司",0,0);
      header.getCellAt(0,0).setColSpan(2);
      header.setCellValueAt("日期：2004-11-8",0,2);
      header.getCellAt(0,2).setColSpan(2);
      return header;
}

/**
 * 创建报表尾
 */
protected CellReportFooter createReportFooter() throws CellReportException {
      CellReportFooter footer = new CellReportFooter(2);
      footer.setCellValueAt("制表人：XXX",0,0);
      footer.getCellAt(0,0).setColSpan(2);
      return footer;
}

/**
 * 创建报表体
 */
protected CellReportBody createReportBody() throws CellReportException {

      CellReportBody body = new CellReportBody();

      // 在表体中添入表体标题
      body.addBlockCellReport(getReportBodyHead());

      // 在表体中添入表体内容
      BlockCellReport bodyContent = getReportBodyContent();
      body.addBlockCellReport(bodyContent);
      return body;

}

/**
 * 创建表体标题
 */
private BlockCellReport getReportBodyHead() {
      // 报表体标题由 2 行 5 列组成
```

```
        CellReportBodyHead bodyhead = new CellReportBodyHead(2, 5);
        // 跨 5 列
        bodyhead.getCellAt(0, 0).setColSpan(5);
        bodyhead.getCellAt(0, 0).setContent("周转  ");
        bodyhead.getCellAt(1, 0).setContent("转入牌号");
        bodyhead.getCellAt(1, 1).setContent("转出牌号");
        bodyhead.getCellAt(1, 2).setContent("调拨率");
        bodyhead.getCellAt(1, 3).setContent("月份");
        bodyhead.getCellAt(1, 4).setContent("年份");
        return bodyhead;
    }

    /**
     * 创建表体内容
     * 从数据库动态创建表体数据
     */
    public BlockCellReport getReportBodyContent() {
        BlockCellReport block = new BlockCellReport();
        Collection col = null;
        try {
            col = CellReportBO_Client.getCellReportBeans(
                    new CellReportBean(), getQueryClause());
            block.addAll((List) col);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return block;
    }

    /**
     * 根据查询对话框，取得查询条件
     */
    private String getQueryClause() {
        Map conditionMap = ui.getConditionMap();
        String condition =
                "select vinpno,voutpno,nflavormny, vmonth,vyear
                  from ycca_phzzb    where vyear = '"
                + conditionMap.get("YEAR") + "' and vmonth= "
                + conditionMap.get("MONTH");
        return condition;
    }
}
```

上面就是一个完整的基于 Cell Report 的报表应用程序，它运行得到的结果是：



这个应用程序展示了 Cell Report 的一部分特性，上面的图示展示了这种报表的组成部分。在取得报表数据这个环节，为了简单起见，我们使用了 Cell Report 自身提供的方式取得后台数据。这是一种理想的方式，但并不实用——在实际业务中，数据模型是 CAVO 数组的居多。不过，这并不造成概念上的冲突，因为在遇到后者这种情况，我们需要自己构造合适的报表块，并决定将哪个数据放入哪个单元格中。

为了更清楚地演示Cell Report报表对单元格和报表块的控制能力，我们稍微修改一个方法：

```
/**
 * 创建表体内容
 * 从数据库动态创建表体数据
 */
public BlockCellReport getReportBodyContent() {
    BlockCellReport block = new BlockCellReport();
    Collection col = null;
    try {
        col = CellReportBO_Client.getCellReportBeans(
            new CellReportBean(), getQueryClause());
        block.addAll((List) col);
    } catch (Exception e) {
        e.printStackTrace();
    }
    // 加入个性化的显示
    BlockCellReport customizeBlock = new BlockCellReport(2,5);
    customizeBlock.setCellValueAt("个性化显示",0,0);
    customizeBlock.getCellAt(0,0).setColSpan(5);
    // 设置背景色
```

```
        customizeBlock.getCellAt(0,0).setBackgroundColor(Color.yellow);
        // 设置对齐方式
        customizeBlock.getCellAt(0,0).setHorizontalAlignment(SwingConstants.CENTER);
        customizeBlock.setCellValueAt("AA",1,0);
        customizeBlock.getCellAt(1,0).setForegroundColor(Color.red);
        customizeBlock.getCellAt(1,0).setColSpan(2);
        customizeBlock.setCellValueAt("BB",1,2);
        customizeBlock.getCellAt(1,2).setForegroundColor(Color.cyan);
        customizeBlock.getCellAt(1,2).setColSpan(2);
        customizeBlock.setCellValueAt("EE",1,4);

        // 将个性化报表块加入更大的报表体中
        block.addBlockCellReportl(customizeBlock);
        return block;
    }
```

界面显示如下：

展示模型个性化报表

	用友软件公司		日期：2004-11-8		
	周转				
	转入牌号	转出牌号	调拨率	月份	年份
	54#	20#	29.86	7	2004
	95#	52#	1436.58	7	2004
	94#	97#	2433.7	7	2004
	94#	54#	693.98	7	2004
	54#	25#	1279.51	7	2004
	15#	94#	772.6	7	2004
	54#	98#	925.98	7	2004
	94#	25#	608.77	7	2004
	54#	35#	2647.61	7	2004
	94#	66#	172.79	7	2004
	54#	95#	9425.89	7	2004
	98#	95#	828.37	7	2004
	98#	35#	188.66	7	2004
	94#	90#	750.94	7	2004
	个性化显示				
	AA		BB		EE
	制表人：XXX				

上面这个例子是为了说明 Cell Report 对报表界面的控制能力，以及动态添加报表块的简单性。利用报表块，我们可以构建很多界面复杂的报表，当然，就像前面所说，这种方法是重量级的，它需要编码定制。

➢ 数据模型离散化报表的开发示例

在这个开发实例中，我们假定一个取数场景：在一个数据容器中包含着 8 个数，我们要将每个数在报表的固定的地方显示。开发步骤如下：

1. 定义公式，利用 JEP API 创建公式表达式类。为了简单起见，我们暂时用 HashMap

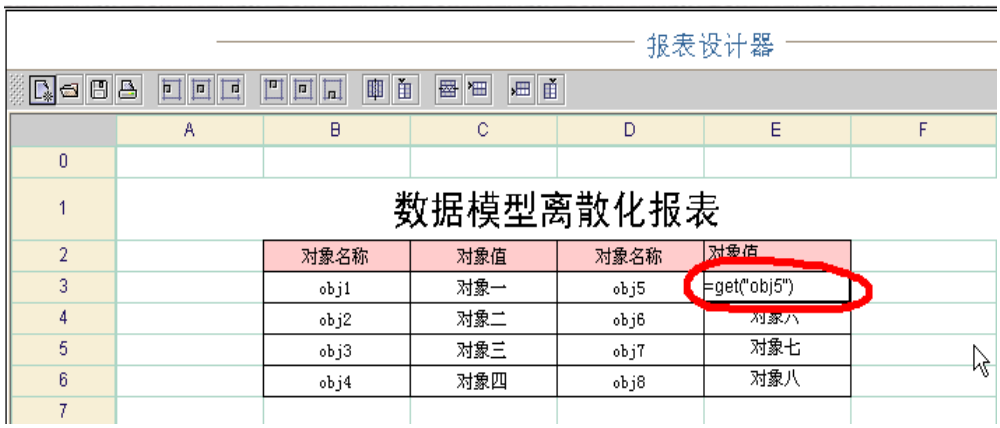
存储这 8 个数，公式公式表达式如下：

```
public class GetValueExpression extends PostfixMathCommand{

    private HashMap m_data;

    public GetValueExpression() {
        numberOfParameters = 1;
        initData();
    }

    private void initData() {
        m_data = new HashMap();
        m_data.put("obj1","对象一");
        m_data.put("obj2","对象二");
        m_data.put("obj3","对象三");
        m_data.put("obj4","对象四");
        m_data.put("obj5","对象五");
        m_data.put("obj6","对象六");
        m_data.put("obj7","对象七");
        m_data.put("obj8","对象八");
    }

    /**
     * 定义如何执行公式
     */
    public void run(Stack inStack) throws ParseException {
        checkStack(inStack);
        Object key = inStack.pop();
        // 从 hashmap 中取得数据
        Object value = m_data.get(key);
        inStack.push(value);
    }
}
```

2. 利用报表设计器，设计报表样式。报表设计器的 UI 类是 CellReportDesignerUI，我们需要修改它，以注册进上面的公式表达式，修改其 registFormula 方法，如下：

```
private void registFormula() {
    JEPExpressionParser parser    =
            (JEPExpressionParser) JEPExpressionParser.getInstance();
    parser.getJepParser().addFunction("get",new GetValueExpression());
}
```

然后，运行 UI。并开始设计报表界面。



将它保存成 **test.report** 文件，放到编译环境 classes 目录下的 jgrid 目录中。

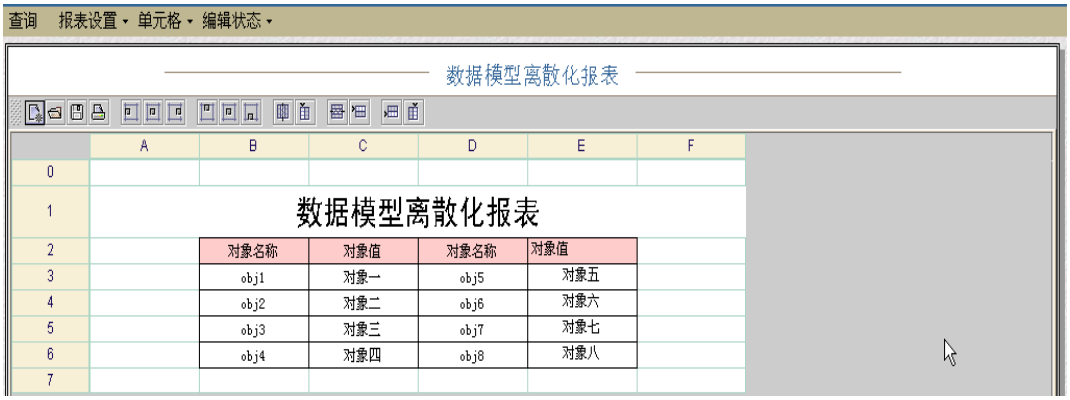
3. 创建报表 UI 类，继承 CellReportUI。如下所示：

```
public class DisperseDataReportUI extends CellReportUI{

    public DisperseDataReportUI() {
        super();
        // 注册公式表达式
        JEPExpressionParser parser =
                (JEPExpressionParser) JEPExpressionParser.getInstance();
        parser.getJepParser().addFunction("get",new GetValueExpression());

        try {
            showReport();
        } catch (CellReportException e) {
            showErrorMessage(e.getMessage());
            e.printStackTrace();
        }
    }

    protected CellReportHandler createCellReportHanlder() throws CellReportException {
        return new DisperseDataReportHandler(this);
    }

    protected AbsractQueryDLG createQueryDLG() {
        return null;
    }

    public String getTitle() {
        return "数据模型离散化报表";
    }
}
```

4. 创建报表处理类，继承 StaticCellReportHandler 类，如下：

```
public class DisperseDataReportHandler extends StaticCellReportHandler{

    public DisperseDataReportHandler(CellReportUI ui) {
        super(ui);
    }

    /**
     * 报表设计文件名称
     */
    public String getDesignFileName() {
        return "test.report";
    }
}
```

5. 界面示例如下：



- **开发需注意的问题**

1) 数据模型离散化报表在使用时，在效率上需要格外注意。因为数据模型的数据往往要从多个数据表按不同查询条件取得，如果每个单元格在执行公式时都要从后台取数，效率问题会非常明显。因此建议此时采取合适的数据访问模式，尽量减少数据库访问次数。
2) 动态行、动态列的报表利用 Cell Report 报表块的概念可以开发出，但是需要对其 API 比较熟悉。
3) 与基于报表模板的方式相比，本节介绍的方法的最大的优点是展示模型很灵活，但它存在着一些功能上的欠缺，比如打印、数据组织等。另外，它是一种重量级的方法，需要开发者做许多编码工作。是否选用该方案，一定要结合应用场景慎重分析。

附录

1．模式一览表

模式大类	模式	行列特性	模式特性	开发含量
查询类报表	1、明细报表	动态行固定列	不含分组统计的查询数据报表	低
	2、汇总报表	动态行固定列	包含分组统计的查询数据报表	低
	3、动态 SQL	动态行动态列	不同查询条件设置要求不同表列结构	中
交叉类报表	4、复合查询	动态行固定列	多个查询结果通过行 ID 连结成一个最终结果	低
	5、CASE-WHEN	动态行固定列	单个查询通过多个 CASE-WHEN 表达式连结不同条件分支的统计结果	低
	6、旋转交叉	动态行动态列	根据指定维度进行真实的数据旋转，交叉后列结构在设计态完全不可知	低
投影类报表	7、投影交叉	固定行固定列	单元格取值由行条件和列条件与固定的查询统计联合决定	低
	8、合并查询	固定行固定列	报表部分区域由查询结果填充，部分区域由手工直接录入	低
算法类报表	9、普通占比	动态行固定列	普通报表要求计算行间或列间占比	中
	10、投影占比	固定行固定列	投影报表要求计算行间或列间占比	中
	11、程序送数	动态行动态列	报表展现数据完全由业务程序提供	高
高级应用	12、普通穿透	——	普通报表选中行穿透到明细	低
	13、投影穿透	——	投影交叉表选中单元穿透到明细	低
	14、主从连动	——	主表选中行刷新子表明细	低
	15、统计图表	——	图形控件展现表格数据	低
行业报表工具	16、标准报表	动态行动态列	报表的每行均由多表连接查询出来的原始数据组成	高
	17、分块填充报表	动态行动态列	同一纵向分块的数据对应同一分类信息，同一横向分块的数据对应同一业务系统信息，块内行顺序不明感	高
CELL 报表工具	18、CELL-REPORT	动态行动态列	数据来自离散数据源，展现具有强烈个性化要求	高

2. 查询引擎 FAQ 之参数篇

1、参数定义中的“操作符提示”列是什么意思？是否有用？
答：通常此列内容只有提示意义而无实际用处。只有对于参照型参数并且所填内容为 like 或 in 的时候有特殊含义——前者会在参照出来的值后面加百分号，后者支持参照多选。

2、数值型参数和字符型参数有何差别？
答：数值型参数保证其在引用过程中参数值不加单引号，而字符型参数可能视引用的具体情况有所不同。

3、在进行复合查询的参数定义时，是否需要列出所引用子查询的参数？
答：是的，复合查询不会自动去递归提取所引用子查询的参数并集。这个并集需要通过手工引用。复合查询自身不一定使用其中的参数，但它负责把参数分发给引用的子查询。

4、如何快速引用其它查询对象的参数？
答：鼠标单击南部面板。

5、什么是替换型参数？
答：参数名以#开头和结尾的参数称为替换型参数（如#param1#），这类参数在引用时被完全字符替换为相应的参数值，而普通参数只限于在向导式设计中作为待定条件的取值。

6、“枚举项”列是用来设置什么的？如何设置？
答：对于枚举型参数，枚举项可以用“可选项 1@可选项 2@……@可选项 n”描述，也可以用“select distinct 字段 from 表”描述。对于参照型参数，枚举项可以选择预置的基础参照，也可以使用程序员自己开发的参照类（参见问题 8），也可以使用自由参照（参见问题 9）。对于字符型参数，除非枚举项中填入了自定义录入界面的类名（参见问题 10），否则枚举项无实际意义，也无须填写。

7、如何让参照型参数的参照支持多选？
答：参见问题 1，即将操作符提示设为“in”。

8、如何使用程序员自己开发的参照模型？
答：在参照型参数的枚举项中用“<全类名>”格式填写参照类名——要求该类继承自 AbstractRefModel，且有无参构造子。

9、如何使用自由参照？
答：假定有一个非复合查询 a，用户可以在另一个查询对象 b 的参数定义中指定一个依赖于 a 的参照型参数，格式为：
[依赖查询的 ID,参照标题,主键列序号,编码列序号,名称列序号]
——比如[a,自己写的参照,0,1,2]
写在枚举项里面，这时浏览查询 b 进行参数设置的时候，对应参数弹出的参照即为自由参照。它突破了参照必须由基础数据预置或者程序员编程的限制。

10、程序员如何自定义参数的录入界面？
答：程序员写一个类继承 nc.ui.pub.querymodel.IAdvParamDlg，实现其中的两个抽象方法，然后把“<全类名>”写到字符型参数的枚举项中，参数设置的时候就能够弹出自定义界面。要求这个界面的返回值要组织成字符串形式。

11、什么是参照依赖？
答：假定参数 1 是公司参照，参数名为 corp，参数 2 为部门参照，参数名为 dept，现在希望 dept 参照出 corp 所选公司下的部门，就可以写参数依赖表达式。格式为“bd_deptdoc.pk_corp=[corp]”，运行时“[参数名]”会被替换为对应的参数取值。

12、参数设置时参数的录入界面有几种？
答：四种——直接录入（对应一般的字符型和数值型参数），下拉框录入（对应字符枚举和数值枚举型参数），参照录入（对应参照型参数），自定义界面录入（参见问题 10）。

13、参数设置时将某个参数值设为空意味着什么？
答：如果是替换型参数，则对应的参数值被替为空字符串；否则，该参数对应的待定筛选条件忽略（相当于 1=1）。

14、浏览格式与浏览查询在参数设置的时候有何差别？
答：后者的设置界面只可能是单页签，前者根据所引用查询别名的数目可能会有多个页签。

15、浏览格式时参数设置界面中的快捷按钮有何作用？
答：快捷按钮只在多页签的设置界面中出现，用于将第一页签的设置应用到其它页签的同名参数。