提效报表开发的通用工具

用集算器高效开发报表

1 概要

报表是很多 web 应用系统不可缺少的模块，仍然是绝大多数 BI 项目的基础功能。随着数据时代的到来，数据来源越来越多样(text,excel,monogdb,redis,es...)，为报表数据准备带来了挑战，传统做法还是先将库外数据到数据库里，再利用数据库的计算能力（写 SQL 或存储过程）为报表准备数据。因依赖前置导入，报表实时性得不到保障，报表开发流程也被拉长，随着报表需求与日俱增，数据库也越来越臃肿，管理成本不断升高。

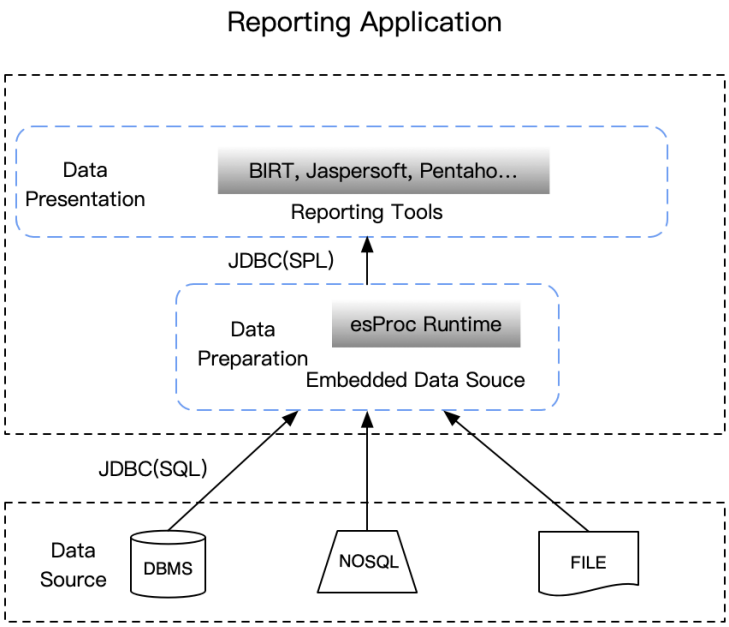
但如果直接使用这些库外数据做报表开发，总是费时费力，最终完成的报表，性能往往也不够好。究其原因，熟悉报表开发的程序员都知道，报表工具内只提供运算这类库外数据的几种简单能力，当计算需求复杂时，需要扩展到报表工具外部，用户自定义方式去实现，这类计算通常用高级语言(Java,.net)硬编码实现，需要很强的数据编程经验，经常超过了普通报表开发人员的能力范围，用高级语言实现的开发成本很高，并且不可复用。

即便数据在数据库里，计算复杂报表经常需要通过好用的高级窗口函数或存储过程才能实现，这些都是开源数据库（mysql,hive…）的短板，商业数据库这方面要好很多，但实现起来也并不轻松，通常都需要用到 SQL 的高级扩展，各数据库厂商对 SQL 扩展都不尽相同，这就需要精通某种数据库高手才能完成。熟悉各种数据库，进阶成 SQL 高手，并非易事，如何让普通报表开发人员，用相同的方式，轻松搞定这类问题呢？另外，存储过程和数据库耦合在一起，大量使用会给数据库运行带来巨大开销，维护也非常麻烦。

如果能有一种计算引擎，具有和数据库相同的计算能力，不必导入数据，直接计算各种来源的数据，提供通用的高级窗口函数和存储过程，独立为报表准备数据，解决上述难题，将会极大提升报表的开发进度和应用效果。

上述内容就是集算器集成版的设计初衷，下面通过介绍其应用结构，举例说明其易用性，结合开源报表工具展示该产品在报表应用场景下的实用价值。

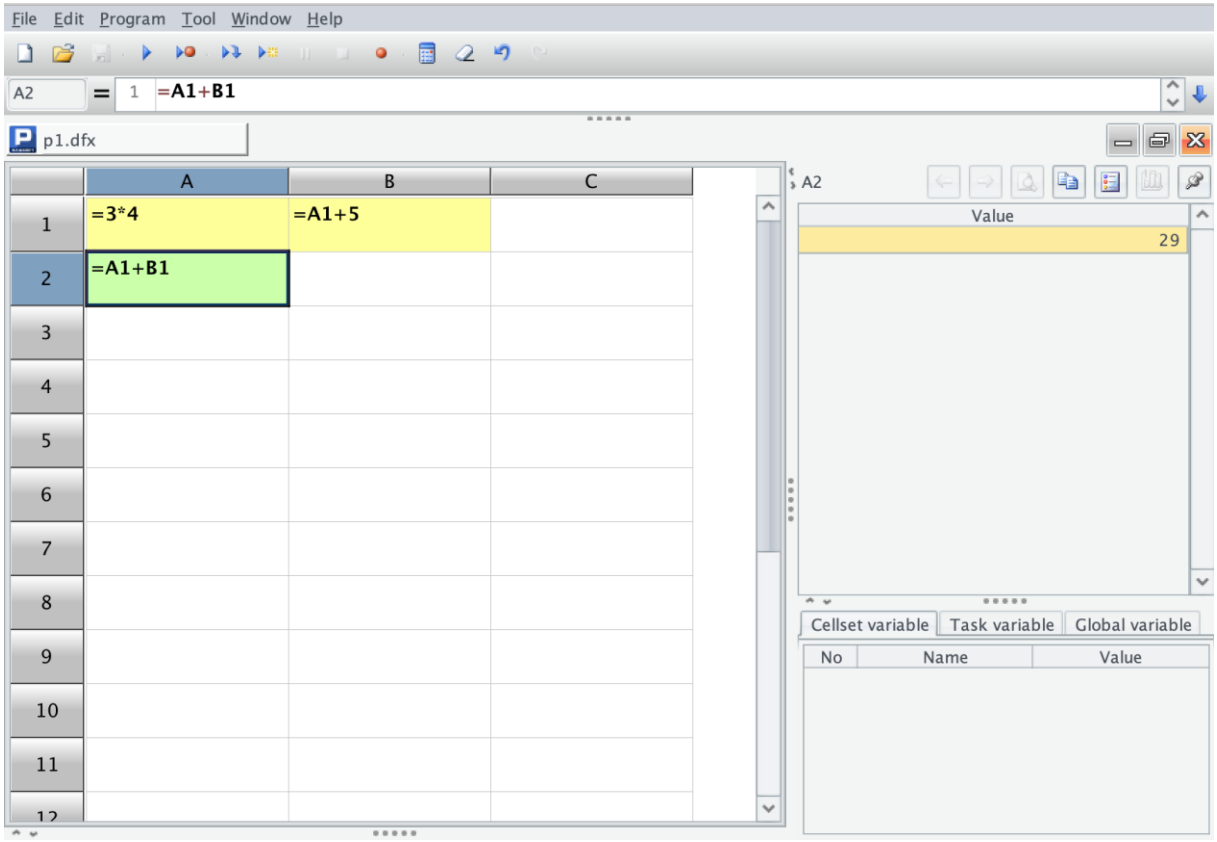
2 应用结构



与数据库作为报表数据源相同，当报表工具嵌入了集算器运行环境，通过 JDBC 接口，集算器同样作为数据源为报表准备数据，只不过连接数据库执行的是 SQL，连接集算器执行的是 SPL（集算器脚本语言）

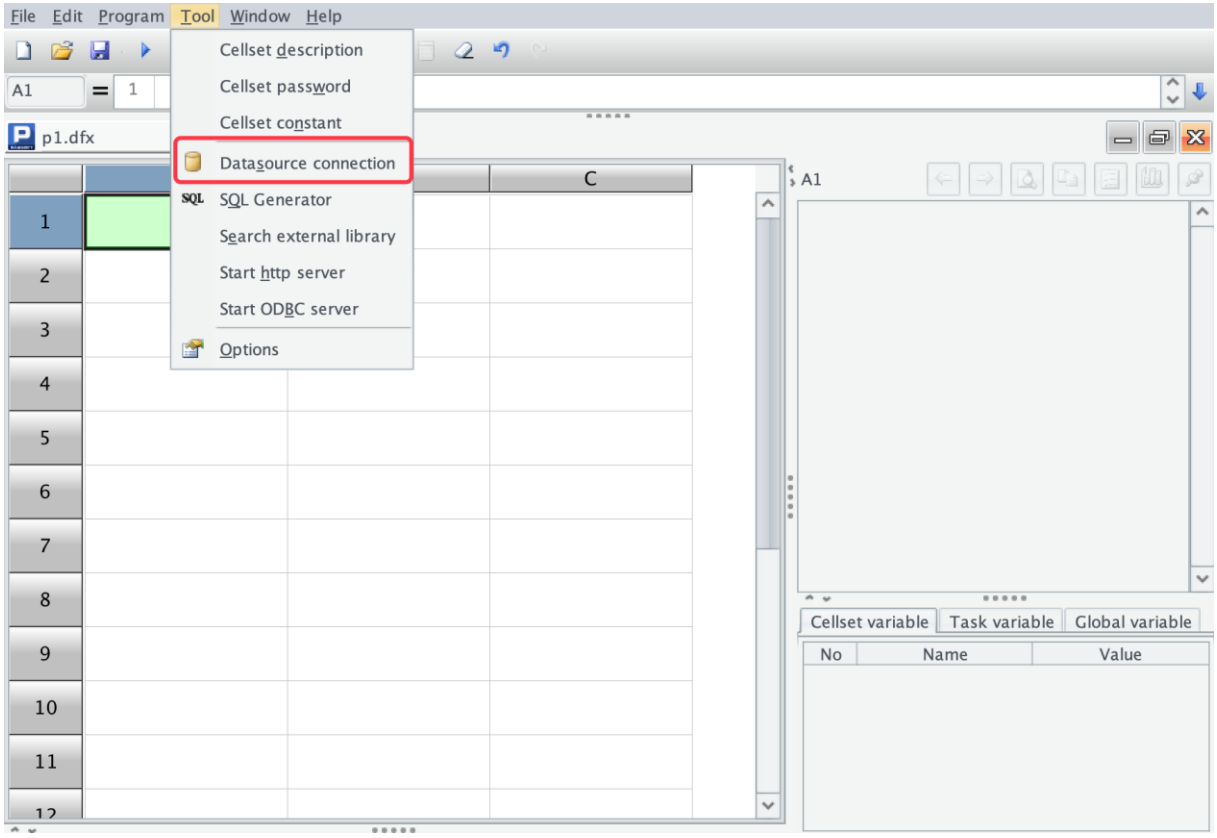
3 开发环境

网格式编程

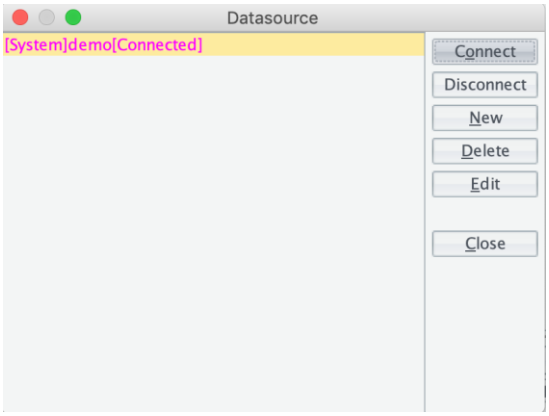


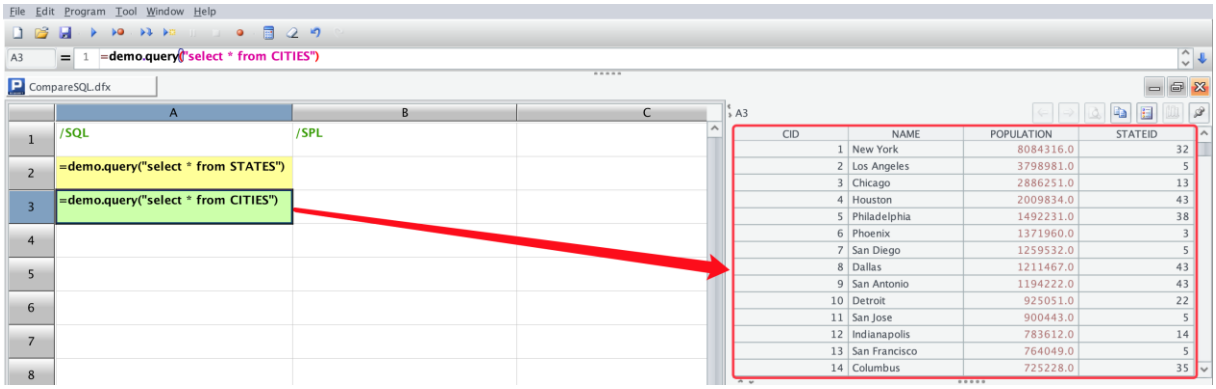
集算器使用网格式编程，和在 Excle 表格里写表达式类似。在单元格里书写集算器脚本 **SPL(Structured Process Language)**，单元格内 SPL 执行的结果赋值给单元格地址，单元格地址作为变量名后续直接引用。代码按单元格顺序，先从左到右、后由上到下。

获取数据

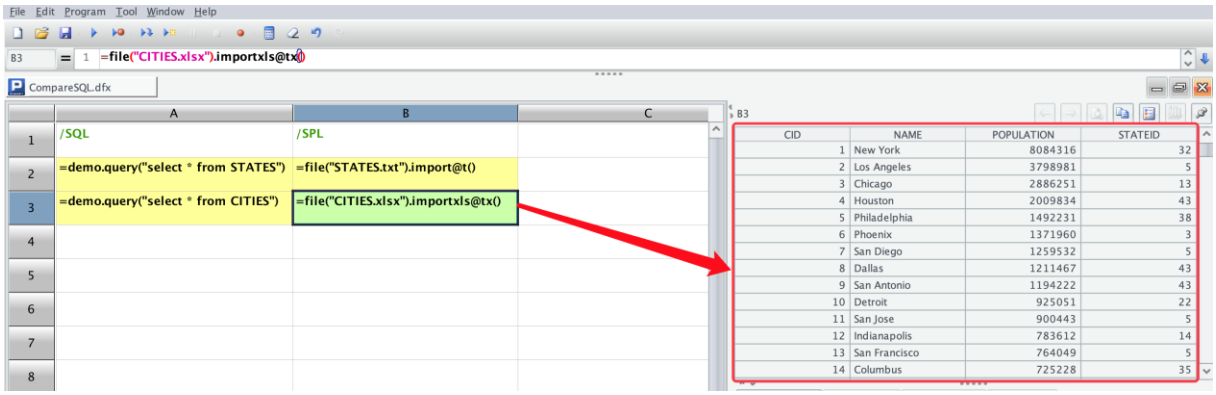


和数据库建立连接（内置的 HSQL 样例数据库）,连接名称为 demo





从 demo 数据库中，取出 STATES 表和 CITIES 表的所有记录。



分别 STATES.txt 和 CITIES.xlsx 是事先从 demo 库里导出的 STATES 和 CITIES 表。除了数据获取的方式不同，A2 和 B2，A3 和 B3 结果都是相同的。一旦数据从各种数据源加载到了集算器，都成为集算器内部的数据对象，使用起来再无任何差别。

SPL 语法

报表开发人员都熟悉 SQL，下面通过几个简单例子，初步了解一下 SPL 语法。

①选出指定字段 select

	A	B
1		=file("STATES.txt").import@t()
2	=demo.query("select NAME as STATE,CAPITAL from STATES")	=B1.new(NAME:STATE,CAPITAL)

从数据表中选出需要的字段，A2 和 B2 中的结果是相同的：

STATE	CAPITAL
Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
Arkansas	Little Rock
Colorado	Denver
Connecticut	Hartford
Delaware	Dover
Florida	Tallahassee
Georgia	Atlanta
Hawaii	Honolulu
Idaho	Boise
Illinois	Springfield
Indiana	Indianapolis
Iowa	Des Moines
Kansas	Topeka

②筛选记录 where

	A	B
1		=file("STATES.txt").import@t()
2	=demo.query("select NAME as STATE,CAPITAL from STATES where POPULATION>15000000")	=B1.select(POPULATION>15000000).new(NAME:STATE,CAPITAL)

A2 和 B2 中选出了人口 15,000,000 以上的州的数据，结果是相同的：

STATE	CAPITAL
Florida	Tallahassee
New York	Albany
Texas	Austin
California	Sacramento

③去重 distinct

	A	B
1		=file("CITIES.xlsx").importxls@tx()
2	=demo.query("select distinct STATEID from CITIES")	=B1.id(STATEID)

A2 和 B2 中获得了 **CITIES** 表中的城市所在各州的编号，结果是相同的：

Member
1
2
3
5
6
9
10
11
12

④排序 order by

	A	B
1		=file("CITIES.xlsx").importxls@tx()
2	=demo.query("select * from CITIES order by NAME")	=B1.sort(NAME)
3	=demo.query("select * from CITIES order by NAME desc")	=B1.sort(NAME:-1)

A2 和 B2 中将数据根据城市名升序排序：

CID	NAME	POPULATION	STATEID
32	Albuquerque	463874	31
64	Anaheim	334425	5
74	Anchorage	278700	2
37	Argentina	435245	5
49	Arlington	349944	43
36	Atlanta	424868	10
38	Auckland	402777	5
68	Aurora	303582	6
15	Austin	671873	43
67	Bakersfield	308392	5
16	Baltimore	638614	20
89	Baton Rouge	229553	18

A3 和 B3 中将数据根据城市名降序排序：

CID	NAME	POPULATION	STATEID
111	Yonkers	197852	32
61	Wichita	357698	16
19	Washington	570898	47
35	Virginia Beach	433934	46
40	Tulsa	391908	36
27	Tucson	503151	3
69	Toledo	298446	35
65	Tampa	332888	9
71	Stockton	290141	5
80	St. Petersburg	248098	9
76	St. Paul	273535	23
45	St. Louis	338353	25

⑤分组汇总 group by

	A	B
1		=file("CITIES.xlsx").importxls@tx()
2	=demo.query("select STATEID, count(*) as CITYCOUNT from CITIES group by STATEID")	=B1.groups(STATEID;count(~):CITYCOUNT)

A2 中的 SQL 分组统计各个州在城市表 **CITIES** 中的城市总数，结果如下：

STATEID	CITYCOUNT
32	4
5	20
13	1
43	13
38	2
3	6
22	1
14	2
35	7
20	1
49	2

B2 中用 **groups** 函数在集算器中分组统计序表（表对象）的数据，其中分号为函数层次分隔符，分隔不同类型的参数（同一类型参数用逗号风格），~和 SQL 中的*类似都是通配符，~抽象代表一行， **count()是聚合函数，**结果如下：

STATEID	CITYCOUNT
1	2
2	1
3	6
5	20
6	4
9	6
10	1
11	1
12	1
13	1
14	2

两种方法获得的汇总结果是相同的，不过集算器中在分组汇总时自动按照分组值完成了排序。

4 报表集成

集算器可以和 BIRT、JasperReport 等报表工具轻松集成，下面以和 BIRT 集成为例，简要介绍一下集成过程，具体操作细节要参考官网相关文档，与其他报表或 BI 系统集成也都类似。

加载 JDBC 驱动

将三个集算器基础 jar 包（集算器[安装目录]\esProc\lib 目录下），拷贝至 Birt [安装目录]\plugins\org.eclipse.birt.report.data.oda.jdbc_4.6.0.v20160607212\driver 下。注意：不同 birt 版本 driver 的上层目录名略有不同

dm.jar	*集算器计算引擎及 JDBC 驱动包*
icu4j_3_4_5.jar	*处理国际化*
jdom.jar	*解析配置文件*

加载部署文件

集算器还有个重要的配置文件 raqsoftConfig.xml（[安装目录]\esProc\config 目录下）， 该文件中配置了授权信息、集算器主路径、dfx 文件寻址路径等各类信息。同样将其拷贝至 Birt [安装目录]\plugins\org.eclipse.birt.report.data.oda.jdbc_4.6.0.v20160607212\driver 下。

打开 raqsoftConfig.xml，配置授权信息。

...

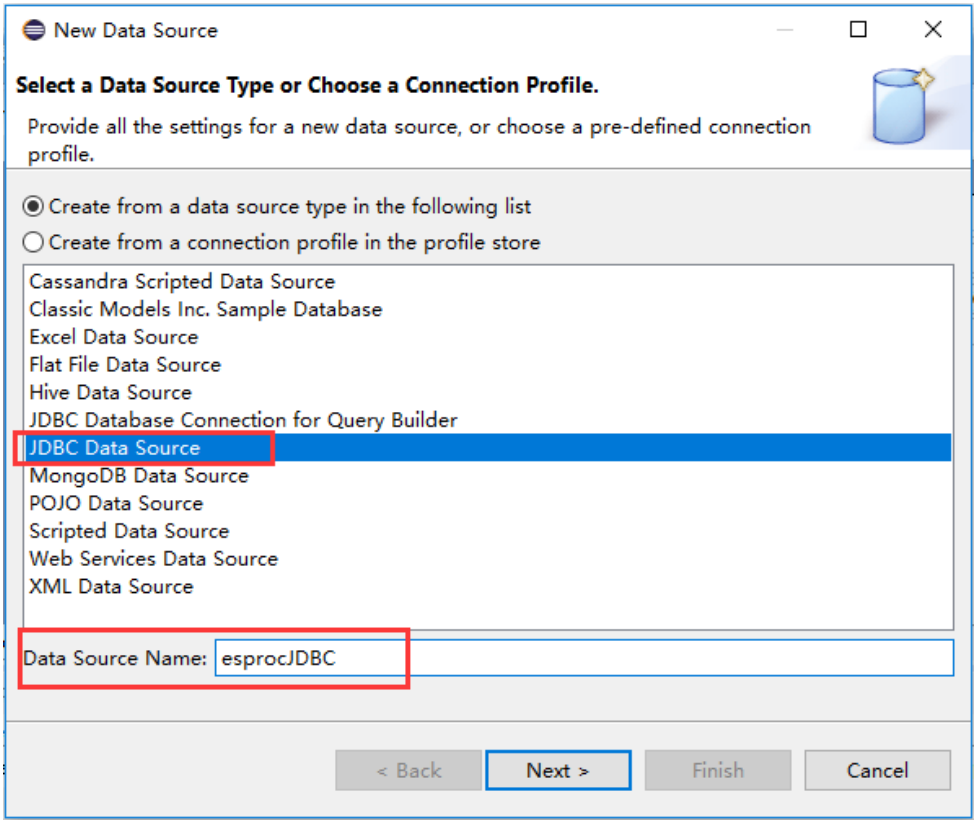
```
<Esproc>
    <license>esproc.xml</license>
    <!-- 商业授权或试用授权，试用授权从官网直接下载-->
    <!-- 授权文件路径，可以是绝对路径，也可以是相对路径，相对路径是相对于类路径-->
```

</Esproc>

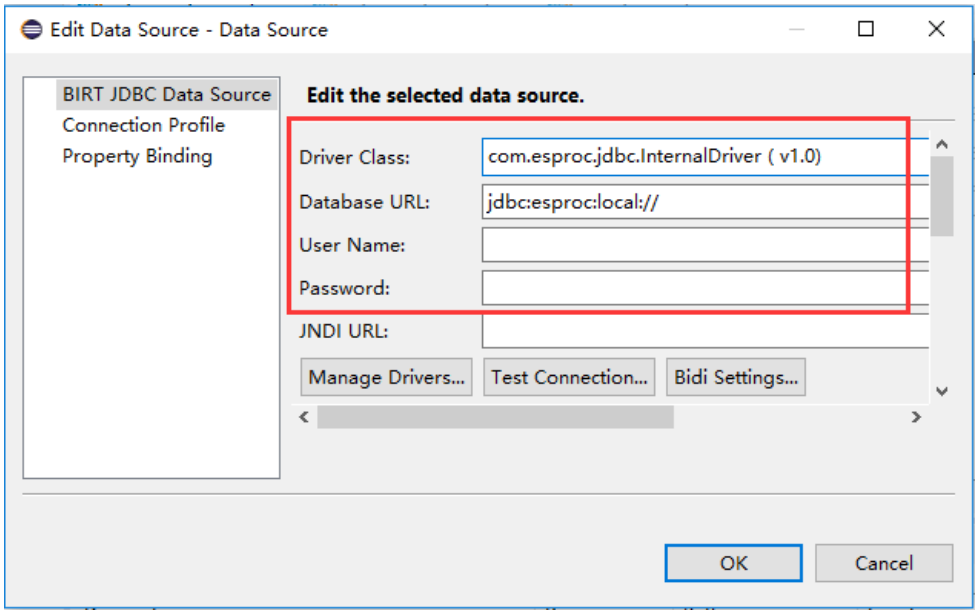
...

新增数据源

新建报表，在“DataSources”下新建 JDBC Data Source 类型数据源，并定义数据源名称为：esprocJDBC。



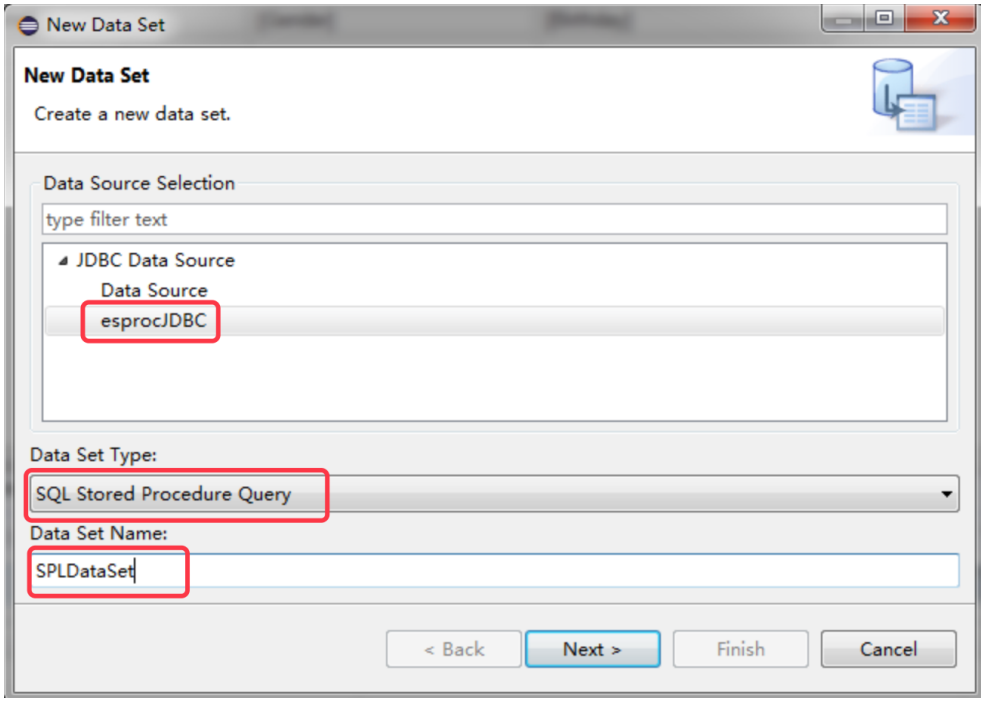
点击【Next】，在 Dirver Class 中选择驱动类名 **com.esproc.jdbc.InternalDriver (v1.0)**，填写 Database URL 为：**jdbc:esproc:local://**，用户名和密码为空。



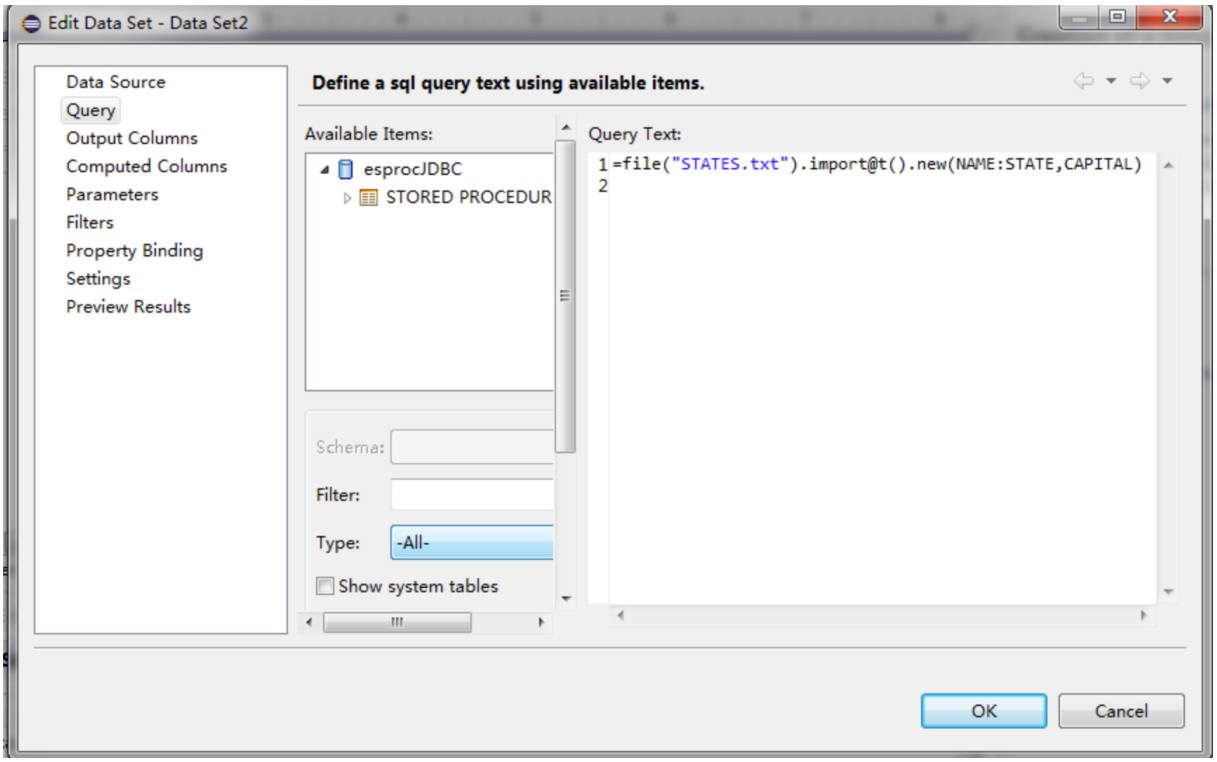
点击【Test Connection】测试连接，出现提示信息：”Connection successful.”则 esprocJDBC 表示连接成功，点击【OK】，数据源创建完成。

新增数据集

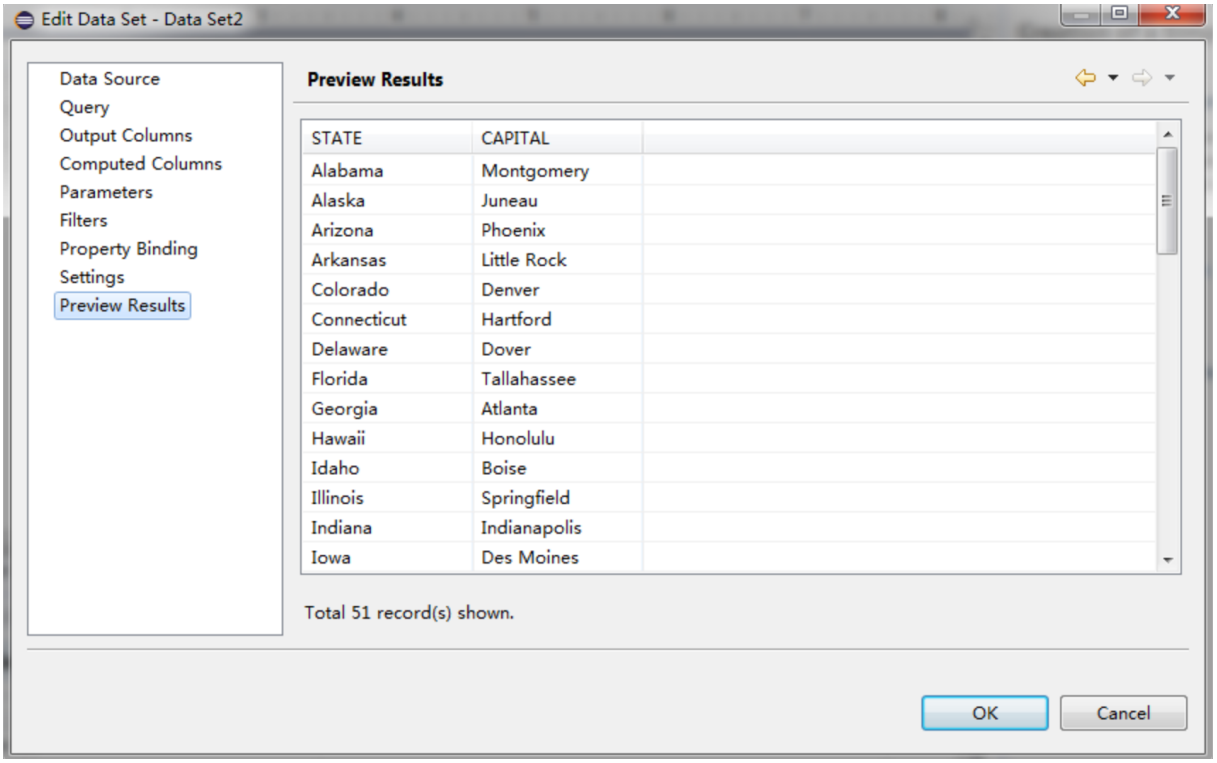
新建“Data Sets”，选择配置的集算器数据源（esprocJDBC），数据集类型选择存储过程（SQL Stored Procedure Query）,数据集名称定义为：SPLDataSet。



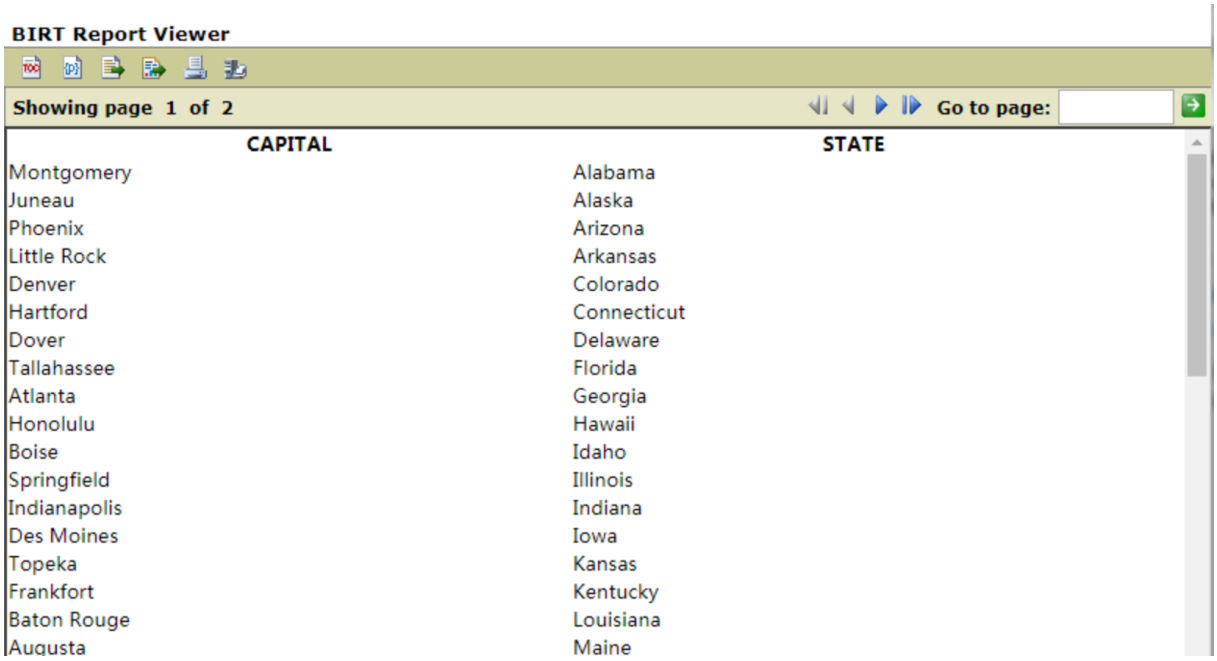
点击【Next】，填写查询语句，将例①中 B1 和 B2 的 SPL 语句合并成一句作为查询语句。



点击左侧的”Preview Results”可预览数据集：



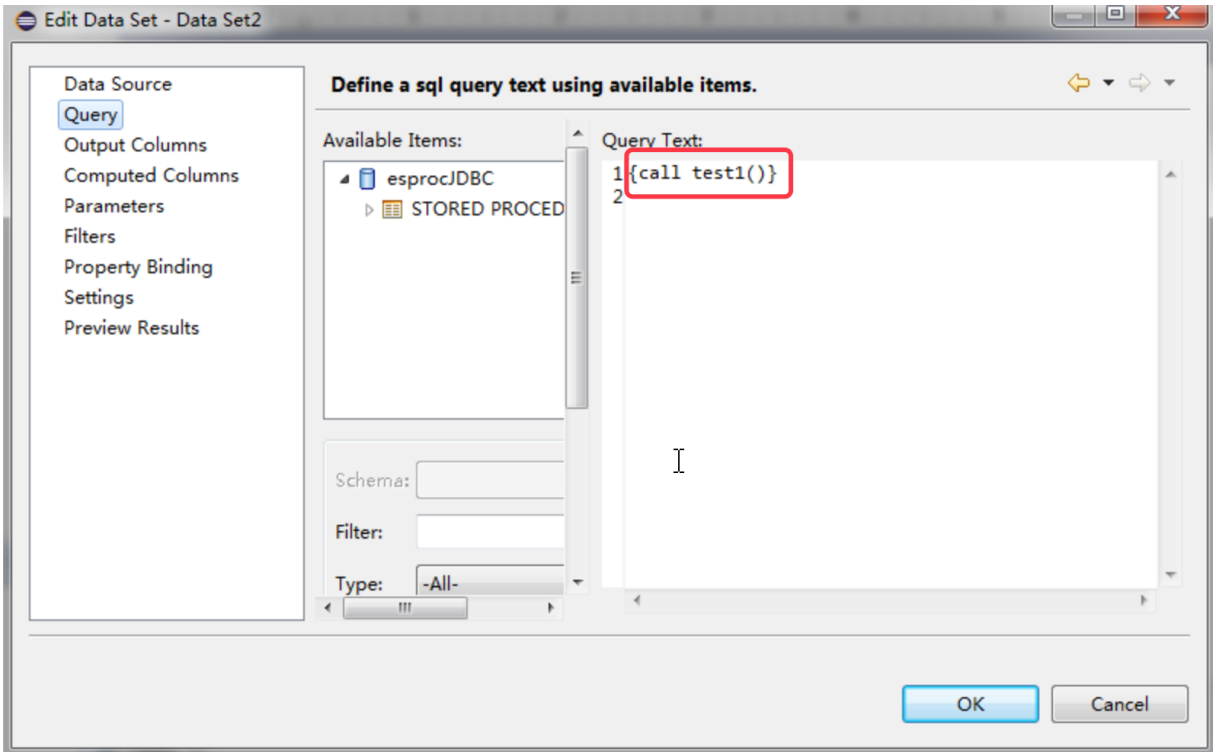
在报表中引用数据集，查看报表：



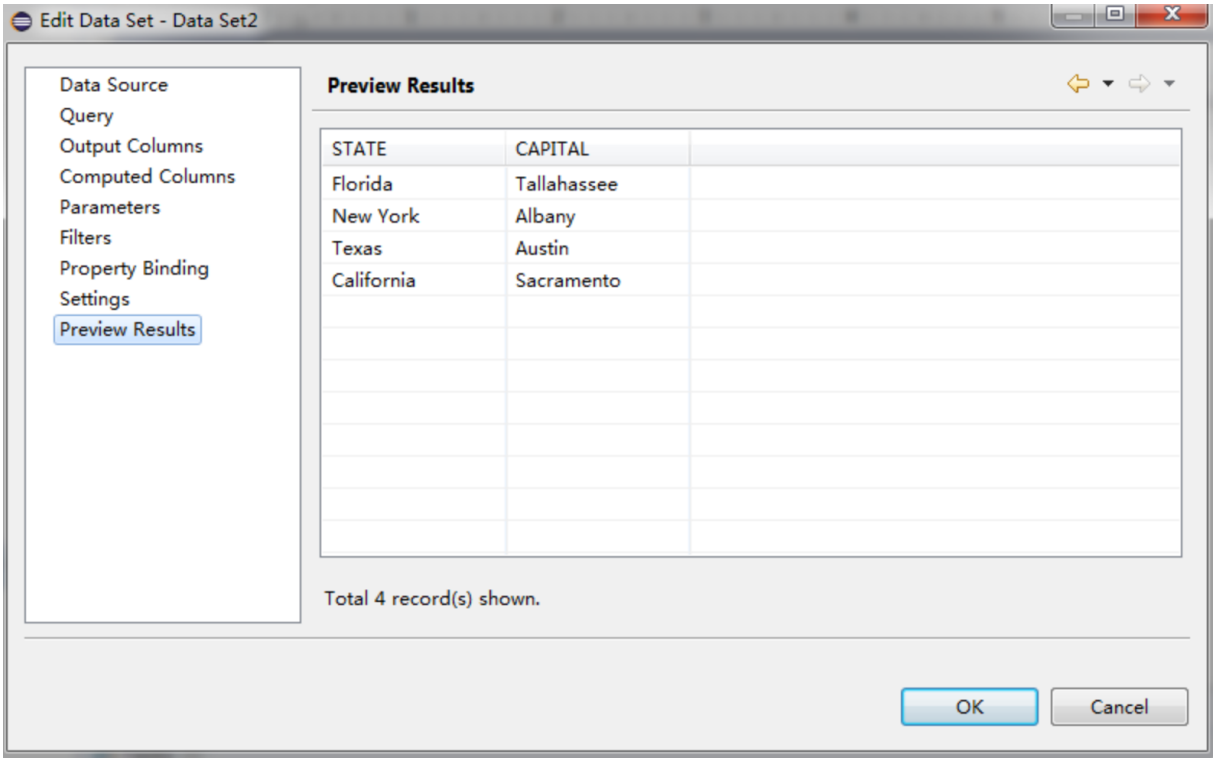
目前，在 BIRT 查询文本框能只能调用一句 SPL，如果想执行多条 SPL，只需用集算器 IDE 将包含多条 SPL 保存，经 IDE 保存的文件扩展名是 dfx。

	A	B
1		=file("STATES.txt").import@t()
2	/=demo.query("select NAME as STATE,CAPITAL from STATES where POPULATION>15000000")	=B1.select(POPULATION>15000000).new(NAME:STATE,CAPITAL)

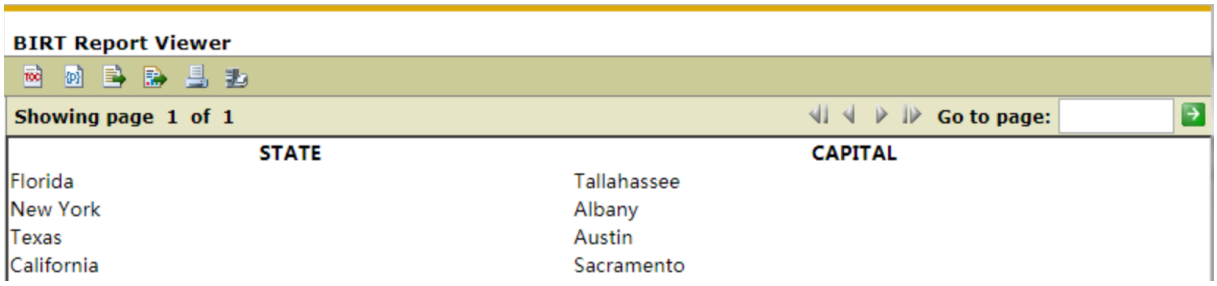
如使用例②，将从数据库取数语句注释掉，只执行从文本取数，然后筛选记录，保存文件为 test1.dfx，复制到部署文件 raqsoftConfig.xml 中<mainPath>C:\work\test_esproc</mainPath>指定的目录下，存储过程里调用脚本文件名 test1 即可。



点击左侧的”Preview Results”可预览数据集，已是筛选后的结果：



在报表中引用数据集，查看报表：



5 应用场景

多样性数据源

许多报表的数据源并不只来源于关系数据库，还可能是 NoSQL 数据库、本地文件、WebService 数据等。在报表工具中，这些非关系数据库的数据源缺乏统一的数据获取接口和语法，有些甚至没有最基本的过滤能力。而计算报表时总还要进行一些过滤甚至关联运算，虽然报表工具也提供一些计算能力，但由于都是内存计算，只适合于数据量较小的情况，数据量较大时就会导致容量负担过重。而且，大多数报表工具也不能很好地处理象 json 或 XML 这种多层数据，也没有灵活编码能力。

如果采用集算器准备数据，则可以直接用这些非数据库数据作为数据源去生成报表，不需要导入数据库的过程，减少开发工作量。

集算器自有的计算能力可以使这些计算能力不一的多样性数据获得通用一致的计算能力，而这将意味着更低的移植成本以及学习难度。

下面举两例说明常见的 json 计算问题。

1、JSON 分组汇总：order.JSON 存储着订单记录，现在要按时间段汇总每个月每个客户贡献的销售额，部分源数据如下：

```
[{OrderID:"26",Client:"TAS",SellerId:"1",Amount:2142,OrderDate:"05-08-2009 00:00:00"},
{OrderID:"33",Client:"DSGC",SellerId:"1",Amount:613,OrderDate:"14-08-2009 00:00:00"},
{OrderID:"84",Client:"GC",SellerId:"1",Amount:89,OrderDate:"16-10-2009 00:00:00"},
{OrderID:"133",Client:"HU",SellerId:"1",Amount:1420,OrderDate:"12-12-2010 00:00:00"},
{OrderID:"32",Client:"JFS",SellerId:"3",Amount:468,OrderDate:"13-08-2009 00:00:00"}…
```

集算器对应的代码：

	A
1	=file("D:\\order.JSON").read().import@j()
2	=A1.select(OrderDate>=argBegin && OrderDate<=argEnd)
3	=A2.groups(month(OrderDate):Month,Client;sum(Amount):subtotal)

将 JSON 文件读为二维表，进行性条件查询，再进行分组汇总，其中 argBegin、argEnd 是参数。结果如下：

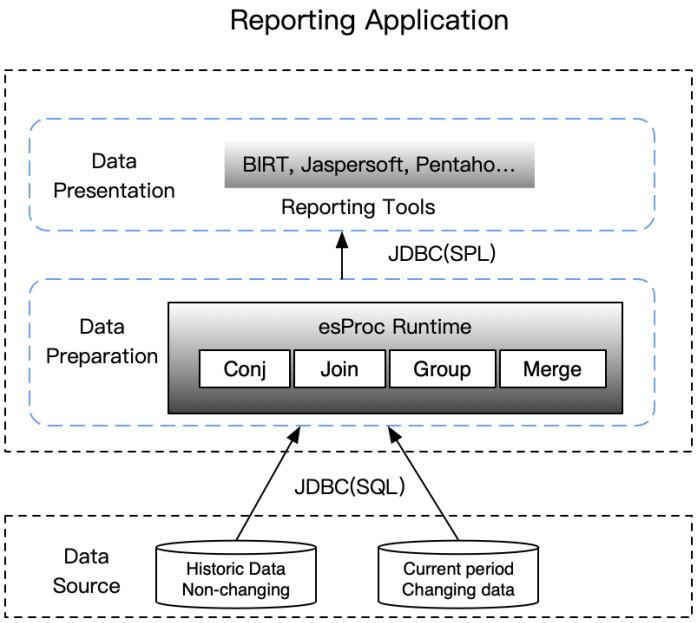
Month	Client	subtotal
7	DY	518
7	GCD	101
7	JDR	1120
7	NR	4031
7	PAER	2491
7	RHD	625
7	WZ	1101
8	DY	1759
8	HP	539

2、有一个 testServlet 可以返回 json 格式的员工信息字符串，需要按条件查询信息，并将结果以 json 形式返回 JAVA 主程序：

	A
1	=httpfile("http://localhost:6080/myweb/servlet/testServlet?table=employee&type=json")
2	=A1.read()
3	=A2.import@j()
4	=A3.select(${where})
5	=export@j(A4)

读取 httpfile 对象，解析 json 格式字符串，生成二维表，按照条件过滤数据，再转换为 json 格式字符串。其中参数 where 是动态的查询条件，形如：BIRTHDAY>=date(1981,1,1) && GENDER=="F"。

跨库混合运算



关系数据库的事务一致性能力目前尚没有有效的替代者，交易系统仍然有必要使用关系数据库来建设。

这种情况下，要实现 T+0 全数据量的实时报表，我们就得把历史数据继续存放在当期的交易数据库中一起计算，历史数据常常要庞大得多，这会要求我们建设更大容量的数据库，成本当然会很高。而且即使愿意支付成本，这个数据量也不可能一直增长，太大了会影响到交易业务的性能，这

就不可容忍了。

通常的办法是把部分历史数据被移出来做个分数据库，这样可以保证交易系统的正常运转，但要实现 T+0 报表就麻烦得多，会涉及到跨库运算。

许多数据库都支持跨库运算，但一般都要求同类型的数据库，但历史数据和当期交易数据的要求不同，数据量更大但不要求事务一致性，很可能使用另一种数据仓库来存储。

而且，即使是同构的数据库，数据库的跨库运算的方法一般也是将另一个库中的数据表映射成本库数据表，实际运算还是一个数据库在做，而且还多出许多数据传递的通讯成本，性能和稳定性都不好。

使用集算器就可以很好地完成这个混合计算任务了。

集算器自己有计算引擎，不依赖于数据库，各个数据库内的数据计算仍由各库进行。集算器可以使用多线程向各数据库同时发出 SQL 语句，由这些数据库并行执行，将各自的运算结果返回到集算器再汇总处理后传给报表工具去呈现。

显然，这种机制还方便横向扩展，历史库可以有多个，是否同类型的也无所谓。

而且，集算器还有服务器方式的集群运行模式，在集算服务器的支持下，历史数据不必一定存放到数据库中，还可以存储在 IO 性能更好的文件系统中，配合集群计算，可以在更低的成本下获得更好的性能。

下面举例说明：

某电信企业用库表 userService 存储用户服务信息，T+0 报表需要呈现各类电信产品的通话时长、通话次数、拨打本地时长、拨打本地次数。实际使用中发现数据量太大，查询效率很低，导致报表性能不佳。

改用计算层可以大幅提升性能，具体做法是将 userService 分地区存储于多个数据库，再用计算层进行并行计算。

以集算器为例，脚本如下：

	A	B
1	**[mysql1,mysql2,mysql3,mysql4]**	
2	**fork A1**	**=connect(A2)**
3		**=B2.query@x("select product_no,sum(allDuration) sallDuration,sum(allTimes) sallTimes,sum(localDuration) slocalDuration ,sum(localTimes) slocalTimes from userService where l0419=? group by product_no", argType)**
4	**=A2.conj()**	
5	**=A4.groups(product_no;sum(sallDuration):ad,sum(sallTimes):at,sum(slocalDuration):ld,sum(slocalTimes):lt)**	

语句 fork 并行开启四线程，每线程从对应的数据库取数，并将分组汇总的结果返回主线程。主线程合并各子线程计算结果，再次执行分组汇总，最终获得报表需要的结果集。

一般报表工具并不具备这种并行取数汇总的能力，需要借助 Java 等高级语言完成，但 Java 并行代码难以书写，缺乏结构化计算能力，比起独立的计算层尚有差距。

库外存储过程

数据库的存储过程本身编写难度并不小，遍历式计算代码的性能也不佳，而且可移植性很差，原则上在报表业务中应当尽量少用存储过程。

采用集算器相当于是算法外置的存储过程，也可以极大程度地减少数据库中的存储过程，算法外置后与报表模板一起存放管理，完全归属于报表模块，不仅降低与应用其它部分的耦合，更不会造成与其它应用的耦合。

不过，集算器并不能完全替代存储过程，某些涉及数据量巨大难以移出库外计算的情况还是可能要用存储过程做些预先处理。

下面举例说明：

1、计算“在每个州的销量均在前 10 名的优秀产品”

库表 stateSales 存储各州各产品的销售信息，有 3 个字段：state，product，amount，记录有重复。需要去除重复，并计算“在每个州的销量均在前 10 名的优秀产品”，最后在报表呈现这些优秀产品的信息。

原先用存储过程计算优秀产品，代码如下：

```
01  create or replace package salesPkg
02  as
03      type salesCur is ref cursor;
04  end;
05  CREATE OR REPLACE PROCEDURE topPro(io_cursor OUT salesPkg.salesCur)
06  is
07    varSql varchar2(2000);
08    tb_count integer;
09  BEGIN
10   select count(*) into tb_count from dba_tables where table_name='TOPPROTMP';
11   if tb_count=0 then
12   strCreate:='CREATE GLOBAL TEMPORARY TABLE TOPPROTMP (
                stateTmp NUMBER not null,
                productTmp varchar2(10)  not null,
                amountTmp NUMBER not null
           )
           ON COMMIT PRESERVE ROWS';
13   execute immediate strCreate;
14   end if;
15   execute immediate 'truncate table TOPPROTMP';
16   insert into TOPPROTMP(stateTmp,productTmp,amountTmp)
    select state,product,amount from stateSales a
     where not(
       (a.state,a.product) in (
         select state,product from stateSales group by state,product having count(*) > 1
       )
       and rowid not in (
         select min(rowid) from stateSales group by state,product having count(*)>1
       )
     )
    order by state,product;
17   OPEN io_cursor for
18   SELECT productTmp  FROM (
       SELECT stateTmp,productTmp,amountTmp,rankorder
       FROM (SELECT stateTmp,productTmp,amountTmp,RANK() OVER(PARTITION BY stateTmp ORDER BY
amountTmp DESC) rankorder
```

```
        FROM TOPPROTMP
       )
   WHERE rankorder<=10 order by stateTmp
   )
 GROUP BY productTmp
 HAVING COUNT(*)=(SELECT COUNT(DISTINCT stateTmp ) FROM TOPPROTMP);
END;
```

改用集算器实现同样的算法则可以避免这些耦合问题。以下是集算器脚本说明：

	A	
1	$select state，product，amount from stateSales	
2	=A1.group@1(state,product)	
3	=A2.group(state)	=A3.(~.rank(amount).pselect@a(~<=10))
4	=A3.(~(B3(#)).(product))	
5	=A4.isect()	

在 A2 去除重复，A3 将数据按州分组，B3 取得每个组排名前 10 的产品的组内序号，A4 按序号取得产品，A5 进行组间交集运算。

如本文在报表集成中讲到的，将上述脚本存为 topPro.dfx 文件，再在报表工具中以 JDBC 的形式调用，调用方法同普通存储过程。调用语句为：call topPro()

上述脚本清晰简单且能解释执行，开发效率高，维护方便。该脚本只需数据库 select 权限，修改和编辑无需额外权限。脚本与报表模板成对存放，便于管理维护。

2、计算“销售额占到一半的前 n 个客户”

有些数据库没有窗口分析函数（eg. Mysql），有些数据库没有存储过程（eg. Vertica），当遇到复杂的数据计算，往往只能通过 Python,R 等外部脚本来实现，但这些脚本语言和主流工程语言（Java）集成性不好，如果直接用工程语言实现类似 SQL 函数和存储过程的功能，经常只是针对某个计算需求编写冗长的代码，代码几乎不可复用。

即便拥有强大的分析函数，实现稍复杂的逻辑其实也不算容易，比如下面这种常见的业务计算，找出“销售额占到一半的前 n 个客户，并按销售额从大到小排序”，在 Oracle 中 SQL 实现如下：

```
with A as
(selectCUSTOM,SALESAMOUNT,row_number() over (order by SALESAMOUNT) RANKING
from SALES)
select CUSTOM,SALESAMOUNT
from (select CUSTOM,SALESAMOUNT,sum(SALESAMOUNT) over (order by RANKING) AccumulativeAmount
from A)
where AccumulativeAmount>(select sum(SALESAMOUNT)/2 from SALES)
order by SALESAMOUNT desc
```

按照销售额累计值从小到大排序，通过累计值大于“一半销售额”的条件，逆向找出占到销售额一半的客户。为了避免窗口函数在计算累计值时对销售额相同的值处理出现错误，用子查询先计算了排名。

下面是用集算器实现相同逻辑的代码：

	A	B
1	=connnect("verticalCon")	/建立数据库连接
2	=A1.query("select * from sales").sort("SALESAMOUNT:-1")	/按销售额从大到小排序
3	=A2.cumulate(SALESMOUNT)	/计算出销售额的累计值序列，此处取代数据库窗口函数
4	=A3.m(-1)/2	/根据最后的累计值，算出“一半销售额”
5	=A3.pselect("~>=A4")	/在累计值序列找到大于“一半销售额”值的所在位置
6	=A2(to(A5))	/找出“一半销售额”值所在位置及其前面的所有记录
7	>A1.close()	/关闭数据库连接
8	return A6	/结果返回

从上述代码我们可以看到，集算器利用一套简洁的语法取代了需嵌套 SQL+ 窗口函数才能实现的逻辑，并且具有通用一致性（任何数据源代码一致）。

SQL 难点解决

分组子集

除了数据表外，SQL 没有显式的集合数据类型，在分组时会强迫计算出聚合值。但有时我们感兴趣的不只是聚合值，还有分组子集，这时 SQL 就很难处理，要用子查询反复计算。

集算器有集合数据，也提供了返回子集的分组函数。这样就能方便地处理分组后运算。

比如找出总分 500 分以上的学生的各科成绩记录。SQL 需要先分组计算出各学生总分，从中过滤出 500 分以上的，再用这个名单与原成绩记录 JOIN 或用 IN 判断，较麻烦且要重复取数。而集算器则可以按自然思路写出来：

	A	
1	=db.query("select * from R")	
2	=A1.group(student).select(~.sum(score)>=500).conj()	

这种分组后却要返回子集明细记录的情况很多，分组聚合是用来实现某种过滤的中间步骤而不是结果。

有时即使是只要返回聚合值，但聚合计算较为特别，难以用简单聚合函数表示时，也需要保留分组子集用于再计算。

这类计算在现实中并不少见，但因为计算复杂，常常涉及较多的业务背景，不适合举例说明，这里改造了一个简化后的例子：

设有用户登录表 L 结构为：user（帐号），login（登录时刻）；现要计算出每个帐号最后登录时刻以及该时刻前三天内的登录次数。

找出最后登录时刻很容易，但如果不保留分组子集时则很难计算出那个时间段登录次数。用 SQL 需要先分组计算出最后登录时间，与原表 JOIN 后过滤相应时间段的记录再次分组汇总，不仅麻烦而且计算效率很低。而使用集算器保留了分组子集则容易实现分步式计算：

	A	
1	=db.query("select * from L")	
2	=A1.group(user;~.max(login):last,~.count(interval(login,last)<=3):num)	

其中~就是按 user 分组后的子集。

如果数据有序还可以用高效的方法计算：

	A	
1	=db.query("select * from L order by login desc")	
2	=A1.group(user;~(1).login:last,~.pselect@n(interval(login,last)>3)-1:num)	

有序聚合

取出每组的前 N 条、最大值对应记录等也是较常见的运算。显然，这些都可以用保留分组子集的方法实现，但由于这类运算较常见，集算器将其理解成某种聚合而提供了专门的函数，这样就可以采用和普通的分组汇总基本一致的处理方式。

SQL 没有集合数据类型，离散性也不好，无法提供返回结果是记录引用集合的聚合函数，这种运算就需要子查询等繁琐的方式，经常还会导致大排序而损失性能。

先看最简单的情况，用户登录表 L 结构为：user、login（登录时刻）、IP-address、…；列出每个用户首次登录的记录。

SQL 可以用窗口函数生成组内排序序号，并取出所有序号为 1 的记录，但窗口函数是在结果集上再计算的，因而必须用子查询再过滤的形式，写法有些复杂。而不支持窗口函数的数据库写起来就会更困难了。

集算器提供了 group@1 方法可直接取出每个分组的第一个成员。

	A	
1	=db.query("select * from L order by login")	
2	=A1.group@1(user)	

这类日志数据经常存在文件中，且已经对时刻有序，用集算器就可以直接取出第一条而不必再排序。数据量大到内存放不下时也可以基于游标实现类似的运算。

股价表 S 的结构为：code（股票代码）、date、cp（收盘价）；计算每支股票最近的涨幅。

计算涨幅涉及到最后两个交易日的记录，使用 SQL 需要用两重窗口函数分别实施组内跨行计算再取出结果的第一行，写法繁琐。集算器提供了 topN 聚合函数，利用集合数据直接返回多条记录作为汇总值参与进一步计算。

	A	
1	=db.query("select * from S")	
2	=A1.groups(code;top(2,-date))	/最后 2 个交易日的数据
3	=A2.new(code,#2(1).cp-#2(2).cp:price-rises)	/计算涨幅

聚合函数并不会先计算出分组子集，而是直接在已有结果上累积，这样可获得更高的性能，而且在数据量大到内存放不下时还可以基于游标工作。

如果数据已有序，则可以更高效地用位置取出相应记录：

	A	
1	=db.query("select * from S order by date desc")	
2	=A1.groups(code;top(2,0))	/直接取前 2 条

3	=A2.new(code,#2(1).cp-#2(2).cp:price-rises)	

取出最大值对应记录、第 1 条最后 1 条等类似计算都是 topN 聚合的特例了。

有序分组

SQL 只提供与次序无关的等值分组，但有时分组的键值并不能在每条记录中找到，而是和记录的次序有关，这种情况，用 SQL 又需要使用窗口函数（或其它更麻烦的手段）制造出序号才能实现。

集算器提供了与次序相关的分组机制，方便用于与连续区间相关的计算。

收支表 B 结构为：month、income、expense；找出连续亏损达三月或以上的那些月份的记录。

	A
1	=db.query("select * from B order by month")
2	=A1.group@o(income>expense).select(~.income<~.expense && ~.len()>=3).conj()

group@o 表示在分组时只比较相邻记录，如果相邻值发生变化则会分出一个新组。这样就可以根据收入支出的比较把收支记录分成赢利、亏损、赢利、…这样的组，然后取出其中亏损且成员不少于 3 的组再合并起来。

还是这个表，希望计算收入最长连续增长了几个月。可以设计这样的分组机制：收入增长时和上月分作一个组，收入下降时则分出一个新组，最后统计组成员的最大值。

	A
1	=db.query("select * from B order by month")
2	=A1.group@i(income<income[-1]).max(~.len())

group@i 将在条件变化时分出一个新组，即收入降低时。

在窗口函数的支持下，SQL 也能实现本例和上例的思路，但写法非常难懂。

区间合并也是常见的有序分组运算。设有事件发生区间表 T 有字段：S（开始时刻）、E（结束时刻）；现在要将这些区间中重叠部分去除后再计算该事件实际发生的总时长。

	A	
1	$select S,E from T order by S	
2	=A1.select(E>max(E{:-1}))	/去除被包含的条目
3	=A2.run(max(S,E[-1]):S)	/去除重叠时间段
4	=A2.sum(interval@s(max(S,E[-1]),E))	/计算总时长
5	=A2.run(if(S<E[-1],S[-1],S):S).group@o(S;~.m(-1).E:E)	/合并有重叠的时间段

这里给了多种目标的处理方法，充分利用了跨行运算和有序分组的特点。SQL 要实现这种运算简单用窗口函数已经做不到了，需要用到很难理解的递归查询。

位置利用

对于有序的集合，有时我们需要直接用序号访问成员。SQL 延用了数学上的无序集合概念，要生成序号再用条件过滤才能访问指定位置的成员，这对许多运算造成很大的麻烦。

集算器采用了有序集合机制，允许直接用序号访问成员，这类运算要方便得多。

比如经济统计中常用到的在众多价格中找出中位数：

	A	
1	=db.query@i("select price from T order by price")	
2	=A1([(A1.len()+1)\2,A1.len()\2+1]).avg()	

位置还可以用于分组。事件表 E 结构为：no（序号）、time、act，动作有开始、结束两种，现在要统计事件持续的总时长，即每一对开始和结束之间的时间之和。

	A	
1	=db.query@i("select time from E order by time")	
2	=A1.group((#-1)\2).sum(interval@s(~(1),~(2))	

#表示记录序号，group((#-1)\2)即将数据每两个分成一组，然后针对每组计算时长再合计即可。

根据位置还能进行相邻跨行引用。设有股价表 S 结构为：date（交易日）、cp（收盘价）；现列出计算出股价超过 100 元的交易日及当日涨幅。

	A	
1	=db.query("select * from S order by date")	
2	=A1.pselect@a(cp>100).select(~>1)	
3	=A2.new(A1(~).date:date,A1(~).cp-A1(~-1).cp:price-rises)	

pselect 函数将返回满足条件的成员位置，使用这些位置就可以方便地计算涨幅，而不必象使用窗口函数时事先计算出所有涨幅再过滤。

有时我们要求分组的结果是连续区间，要补齐中间缺省的空子集。SQL 实现这个过程很麻烦，要手工先造出连续不断的分组区间再 left join 要统计的数据表，子查询将不可避免。集算器则可以直接利用位置来实现对位分组。

简化的交易记录表 T 结构为：no、date、amount。现需要按周统计累计的交易金额，没有交易记录的周也要列出。

	A	
1	=db.query("select * from T order by date")	
2	>start=A1(1).date	
3	=interval(start,A1.m(-1).date)\7+1	计算总周数
4	=A1.align@a(A2,interval(start,date)\7)	按周分组，可能有空集
5	=A4.new(#:week,acc[-1]+~.sum(amount):acc)	汇总并计算累计

动态列计算

彻底的集合化，还包括在列上的集合运算。

SQL 是解释执行的语言，可以临时生成动态的数据结构，这方面问题不大。不过，SQL 认为列是数据的属性，是静态的，没有提供针对列上的集合运算。这样在我们事先不知道列的信息或列很多需要通用方式处理时就会显得很麻烦。

列间统计

体育测验表 PE 结构为：name、100m、1000m、long-jump、high-jump、…；成绩等级分为

A、B、C、D 四档，现在要统计各等级在所有项目上的人数合计。

思路很简单，把各项目成绩合并起来再分组汇总即可。SQL 要用大长串的 union 合并各项目，写起来很繁琐。而且列数不确定时还要从数据库中动态获取列名拼接，更为复杂。

集算器支持列上的集合运算，全动态写法轻松简单：

	A	
1	=db.query("select * from PE")	
2	=A1.conj(~.array().to(2,))	从第 2 字段的各项目成绩合并起来
3	=A2.groups(~:grade;count(1):count)	分组汇总

通用转置

对于简单的静态转置，某些数据库提供了 pivot 和 unpivot 语句实现，不支持这些语句的数据库也可以用较繁琐的条件表达式和 union 语句写出来。但转置结果的列由行变换而来，经常是动态的，这时就需要用先算出目标列和行，然后动态拼出另一句 SQL 来执行，但 SQL 不提倡分步运算，实现这种运算即繁琐又难理解。

学生成绩表 R 结构为 student、semester、maths、english、science、pe、art、…，需要双向转置为 student、subject、semester1、semester2、…。

对于简单转置，集算器也提供了 pivot 函数：

	A	B	C
1	=db.query("select * from R")		
2	=A1.pivot@r(student,semester;subject,score)		
3	=A2.pivot(student,subject;semester,score)		

A2 将列转成行，A3 再将行转成列，从而实现双向转置。

还可以采用代码稍复杂但理解起来更简单的通用转置方案：

	A	B	C
1	=db.query("select * from R order by student,semester")		
2	=create(student,subject,${A1.id(semester).string()})		
3	for A1.group(student)	for 3,A1.fno()	=A3.field(B3)
4			>A2.record(A1.student|A2.fname(B3)|C3)
5	return A2		

A2 中先用宏生成目标结果集，再在 A3-C4 的循环中将数据变换后插入到结果集，这是集算器实现转置任务的标准流程，分步运算使代码更清晰易懂。这个方案也可用于静态或单向转置，代码更简单。集算器的列访问机制和动态语言的灵活性，使得各种转置，静态或动态、行转列还是列转行以及双向同时，都可以采用一致的方案实现。

转置计算

设有帐户状态变化表 T：

seq	acount	state	date
1	A	over	2014-1-4
2	A	OK	2014-1-8
3	A	lost	2014-3-21
…			

需要输出指定月份帐户每日的状态，若当日无记录，则延用前一日的状态：

account	1	2	3	4	5	6	7	8	9	…	31
A				over	over	over	over	OK	OK	…	OK
...											

严格地说，这是个静态转置，但列数较多且有规律，完全静态写出很麻烦。而且转置过程中还涉及到列间有序计算，即使有 pivot 语句用 SQL 也难以写出。

而采用集算器则仍然按上述的流程即可简单实现：

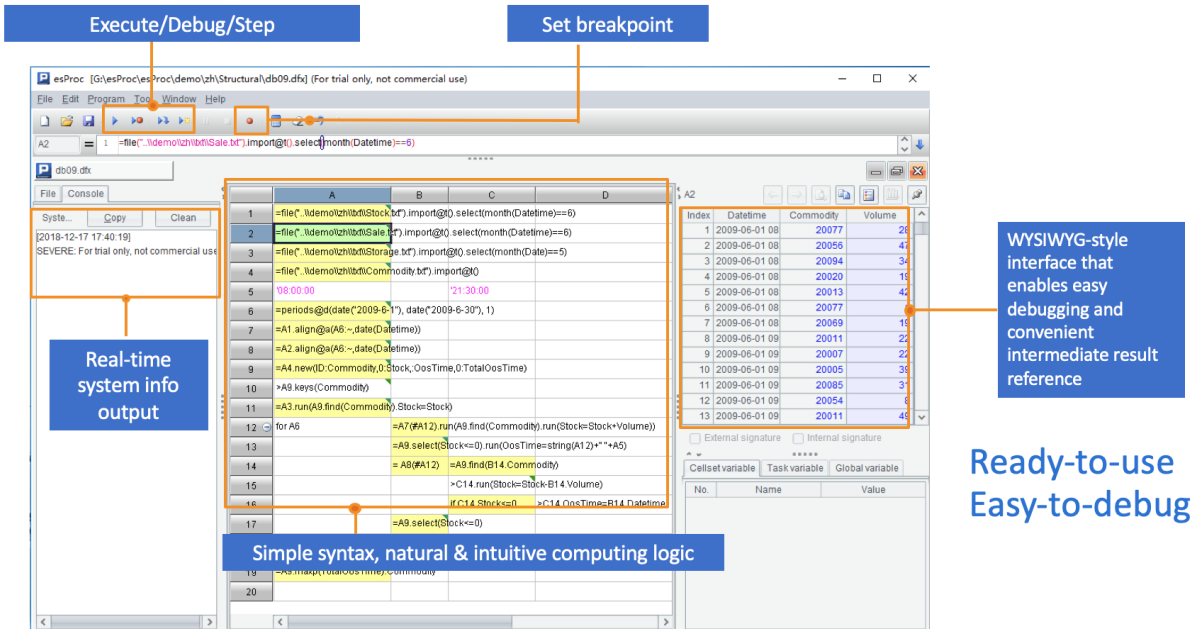
	A	B
1	=db.query("select * from T where year(date)=? and month(date)=?",2014,1)	
2	=create(account,${to(31).string()})	
3	for A1.group(account)	=31.(null)
4		>A3.run(B3(day(date))=state)
5		>B3.run(~=ifn(~,~[-1])
6		>A2.record(A3.account|B3)
7	return A2	

这里只涉及单向转置，比上例少一层循环，B3-B5 中按规则计算插入数据的过程稍复杂些，但整体过程并无不同。

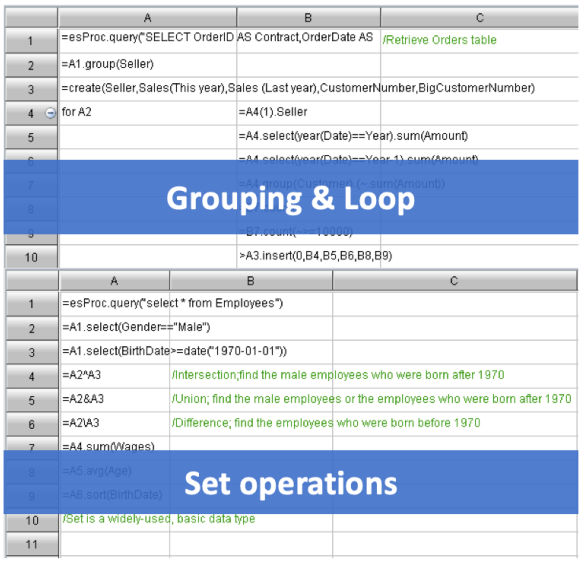
6 易用性

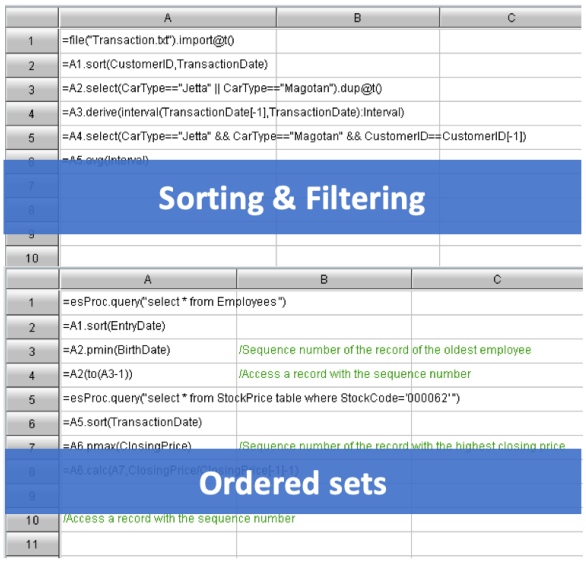
集算器 SPL 是专门为处理结构化数据设计的 DSL(Domain Specific Language)，不像通用语言，目标范围涵盖一切，通过短时间练习就能轻松掌握。分步式处理、中间结果引用，面对复杂需求比 SQL 更容易实现。数据结构简单、语法简洁，比 Python 更容易学习，更容易使用。

界面简洁，调试轻松



语法简单，易学易用





和 SQL 对比

例 1：

To find clients whose sales amount are listed in the top 10 every month

```
select Client from(
   select * from(
      select B.*,row_number() over(partition by month order by SumValue desc)
      rown from(
         select to_char(SellDate,'mm') month,Client,sum(Quantity*Amount) SumValue
         from contract
         where SellDate>=to_date('2012-01-01','yyyy-mm-dd')
         and SellDate<=to_date('2012-12-31','yyyy-mm-dd')
        group by to_char(SellDate,'mm'),Client order by month,client
     ) B
   )C
   where rown<=10
)D
group by Client
having count(Client)=(
    select  count(distinct(to_char(SellDate,'mm')))
    from contract
    where SellDate>=to_date('2012-01-01','yyyy-mm-dd')
    and SellDate<=to_date('2012-12-31','yyyy-mm-dd')
)
```

```
$select SellDate,Quantity,Amount,Client from contract
  where SellDate>=to_date('2012-01-01','yyyy-mm-dd')
  and SellDate<=to_date('2012-12-31','yyyy-mm-dd')
=A1.group(month(SELLDATE))
=A2.(~.groups(CLIENT;sum(QUANTITY*AMOUNT):SumValue))
=A3.(~.sort(-SumValue))
=A4.(~.select(#<=10))
=A5.(~.(CLIENT))
=A6.isect()
```

SQL：
The script is difficult to read and maintain, and can not be simplified via step by step process

SPL：
The script is done in the cells and natural step by step computing is realized by mutual cell-name reference

例 2：

To find out the salesmen whose sales volume have increased by 10% in three consecutive months

```
WITH A AS
    (SELECT salesMan,month, amount/lag(amount)
       OVER(PARTITION BY salesMan ORDER BY month)-1 rising_range
       FROM sales),
   B AS
      (SELECT salesMan,
         CASE WHEN rising_range>=1.1 AND
            lag(rising_range) OVER(PARTITION BY salesMan
                ORDER BY month)>=1.1 AND
             lag(rising_range,2) OVER(PARTITION BY salesMan
                ORDER BY month)>=1.1
          THEN 1 ELSE 0 END is_three_consecutive_month
   FROM A)
SELECT DISTINCT salesMan FROM B WHERE is_three_consecutive_month=1
```

	A	B
1	=sales.group(salesMan).(~.sort(month))	
2	==A1.select(??)	=0
3		=~.pselect(B2=if(amount/amount[-1]>=1.1,B2+1,0):3)>0
4	=A2.(salesMan)	

SQL：
Some computing has to be done by sub query due to lack of set computing.

SPL：
Complex computing can be greatly simplified with explicit set, relative position, sequence reference and computing after grouping, etc.

例 3：

To get the monthly contract growth rate of each salesman

```
with A as(
    select actualSale,Quantity*Amount sales,sellDate from contract
),B as(
   select actualSale,TO_NUMBER(to_char(SellDate,'yyyy')) year,
   TO_NUMBER(to_char(SellDate,'mm')) month,
    sum(sales) salesMonth
   from A group by actualSale,TO_NUMBER(to_char(SellDate,'yyyy')) ,
   TO_NUMBER(to_char(SellDate,'mm'))
   order by actualSale,year,month
),C as(
   select actualSale,year, month, salesMonth,
   lag(salesMonth,1,0) over(order by actualSale,year,month) prev_salesMonth,
   lag(actualSale,1,0) over(order by actualSale) prev_actualSale
   from B
)
select actualSale,prev_actualSale,year, month,
                    (case when prev_salesMonth!=0 and
                     actualSale=prev_actualSale
                     then ((salesMonth/prev_salesMonth)-1)
                     else 0
                     end)LRR
from C
```

	A
1	$select actualSale,Quantity*Amount as sales,sellDate from contract where sellDate>=? and sellDate<=?;startTime,endTime
2	=A1.group(actualSale)
3	=A2.(~.groups(year(sellDate):year,month(sellDate):month; sum(sales):salesMonth))
4	=A3.(~.derive(salesMonth/salesMonth[-1]-1:rate))

SQL：
Disordered data. Ordered computing can only be achieved by indirect script.

SPL：
Totally supports ordered computing, and can easily solve order related problems, like sorting, ranking, top N, comparing with the previous or the same period, etc.

和 Python 对比

例 1：

Sampling(Divide the data into 30% and 70% randomly)

```python
import pandas as pd

data = pd.read_csv("EMPLOYEE_nan.txt",sep="\t")

row_no = pd.Series(range(data.shape[0]))

per_30_no = row_no.sample(frac=0.3)

per_70_no = row_no[~row_no.isin(per_30_no)]

data_per_30 = data.iloc[per_30_no,:]

data_per_70 = data.iloc[per_70_no,:]

print(data_per_30)

print(data_per_70)
```

Time consuming：67ms

Python：
Need to show introducing class libraries, Set operations (difference sets) are slightly more complex. To view the results, you need to print them on display.

	A
1	=file("EMPLOYEE.txt").import@t()
2	=A1.sort(rand())(to(A1.len()*0.3))
3	=A1\A2

Time consuming：6ms

SPL：
Provides rich built-in libraries for direct use, Set operations are very simple to write. The results of each step can be viewed in the results panel.

例 2：

Data conversion(Number fields remain unchanged, and other fields are converted to numbers)

```python
import pandas as pd
import datetime
import numpy as np
import random

data = pd.read_csv("EMPLOYEE.txt",sep="\t")
class_mapping = {}
columns = data.columns
for column in columns:
    try:
        data[column] = data[column].apply(pd.to_numeric)
    except ValueError:
        lg = list(data.groupby(column))
        class_mapping = {}
        for i in range(len(lg)):
            class_mapping[lg[i][0]]=i+1
        data[column]=data[column].map(class_mapping)

print(data)
```

Time consuming：180ms

	A	B
1	=file("EMPLOYEE.txt").import@t()	
2	=A1(1).array().pselect@a(!ifnumber(~))	
3	for A2	=A1.group(~.field(A3))
4		>B3.run(~.field(A3,B3.#))

Time consuming：15ms

Python：
The implementation is relatively simple, but the codes are not short. Identifying blocks of code by indentation is a challenge for users.

SPL：
The overall codes are very short, and the loop functions provided can easily traverse the data. Code blocks are clearly identified through the grid.

例 3：

Generate PivotTable (music as index, user as field, score as score)

```python
import pandas as pd

user_watch_data = pd.read_csv('user_watch1.csv')
music_meta_data = pd.read_csv('music_meta1.csv')
u_i_watch_list = []
gr = user_watch_data.groupby(['user','music'])
for (userid,musicid),group in gr:
    u_i_watch_list.append([userid,musicid,group['listen_time'].mean()])
data_listen_time = pd.DataFrame(u_i_watch_list,
                                columns=['user','music','listen_time'])
data_merge = pd.merge(data_listen_time,music_meta_data,on='music')
data_merge['score'] = data_merge['listen_time']/data_merge['long']
u_i_s_data = data_merge.loc[:,['user','music','score']]
u_i_s_data = pd.pivot_table(u_i_s_data,index = 'music',
                            columns='user',values='score')

print(u_i_s_data)
```

Time consuming：23ms

Python：
Explicit "for" is needed to implement grouping and transpose actions .

	A
1	=file("user_watch1.csv").import@t(;,"")
2	=file("music_meta1.csv").import@t(;,"").keys(music)
3	=A1.groups(user:user,music:music;avg(listen_time):listen_time)
4	=A3.join(music,A2,listen_time/long:score).new(user,music,score)
5	=A4.id(user).concat@cq()
6	=A4.pivot(music;user,score;${A5})

Time consuming：1ms

SPL：
The process code is more concise, without any "for" .

7 联系我们

集算器是一套运行在 JVM 上专门处理结构化数据的中间件，通过 JDBC 接口能轻松和 Java 应用集成，为应用提供可移植、功能强大的计算逻辑。历经 7 年的研发和 3 年的商用，在中国已得到了广泛的应用，和一些国际用户的认可。

FLEXGROUP

Hello raqsoft,

We use your product in the Finance industry, especially in group insurance for Cost Control Monitoring.
We do BI with EsProc to connect with multiples databases such as Vertica, MySql, MS Acces, etc...
The best use for us is to pass parameters to the Vertica database.
We are using Birt (Eclipse) as a reporting tool. EsProc is good at manipulating the different media's used in our reporting tool. It is easy to use, it's like an Excel mix with Sql query that have multiple cells.
Each cell becomes a data array that are easy to use, compare and manipulate. It is very logical and you have made it user friendly.
EsProc is a powerful tool to manage large amounts of data, as all the information can be saved in the internal memory of the server.
Our main concern, was to setup all product together.
EsProc made it simple to integrate the data management with our reports.

Steeve Roy
Vice President, Information technologies
FlexGroup, Canada

2019 年，我们开始拓展国际市场，发展合作伙伴。如果你是数据行业的开发商或分销商，对集算器感兴趣，可以通过官网(www.raqsoft.com)联系我们。同时，如果你是工程师，有报表计算的疑难问题，欢迎来产品社区咨询(c.raqsoft.com)，强大产品的背后，其实是业界最优秀的专家，他们 œ 发现数据开发过程中并没有非常好用的工具，才打造了集算器，对各类数据计算问题有着丰富的经验和通透的理解。