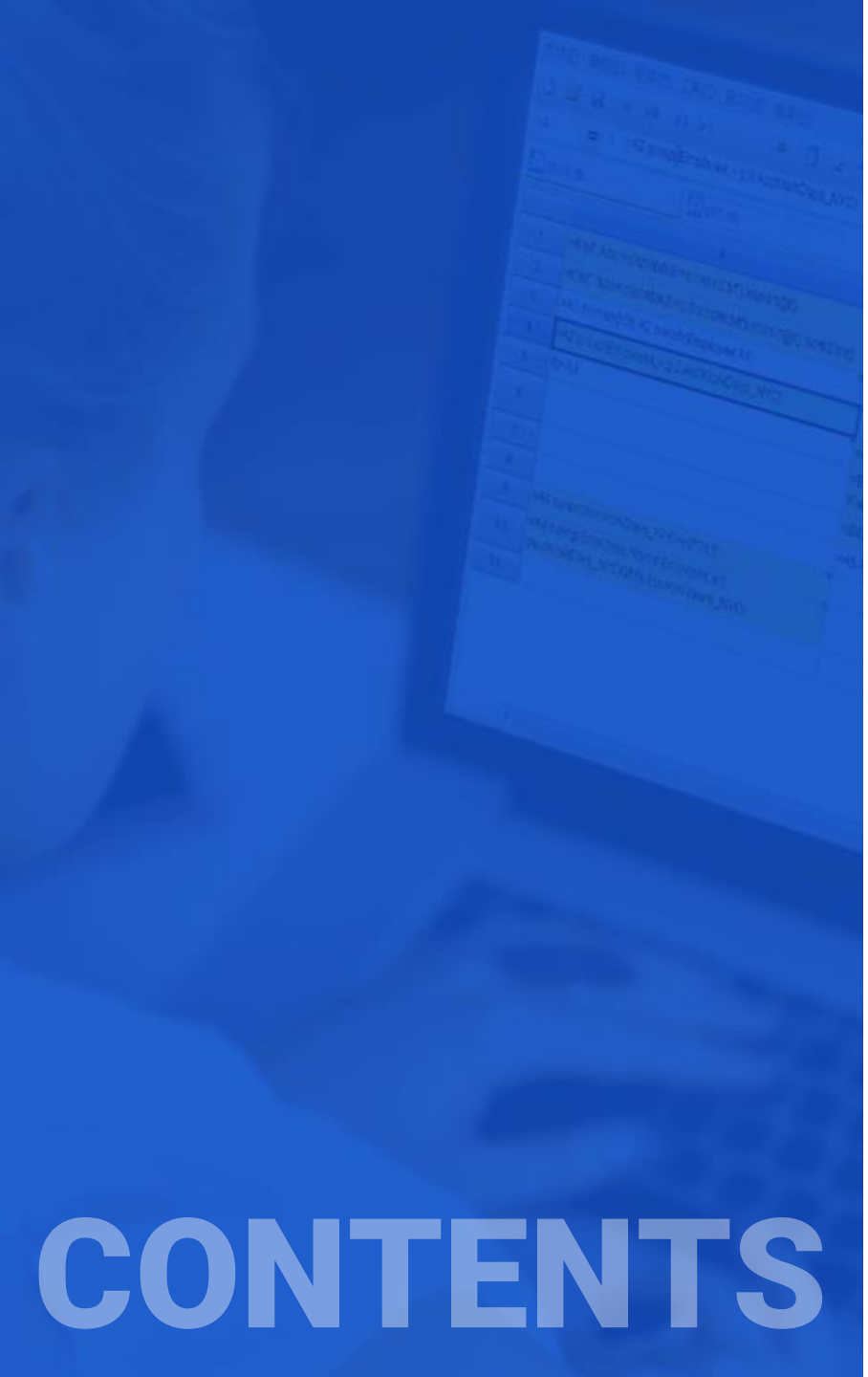
esProc SPL

数据分析引擎

低代码、高性能、轻量级、全功能



CONTENTS

01

esProc是什么

esProc SPL 是什么?

- 数据计算和处理引擎，可用作分析型**数据库**和**中间件**
- 结构化和半结构化数据计算处理
- 线下跑批、在线查询
- 既**非SQL体系**，也非NoSQL技术
- **自创SPL语法**，简洁高效

SPL: Structured Process Language

esProc SPL 应对什么痛点?

面向**线下跑批、在线查询**等数据计算场景

- 时间窗口不够，半夜跑批跑不完，出错来不及重来；月末年头担惊受怕
- 出个报表十分钟，业务人员拍桌子；预计算难预测，业务人员不满意
- 在线用户多一点，时间跨度长一点，数据库就像死了一样
- N层嵌套长SQL，存储过程几十K，过几天自己都看不懂
- DB/NoSQL/文本/Json/Web几十种数据源，做梦都想跨源混合算
- 数据量大了冷热数据分库，再想全量（T+0）统计难死人
- 过度依赖存储过程，应用难移植，架构难调整
- 数据库里表太多，存储计算资源耗尽，想删不敢删
- 报表没完没了做不完，人员成本投入何时休
- ……

esProc SPL 对标什么?

采用SQL语法的、应用于OLAP场景的数据库

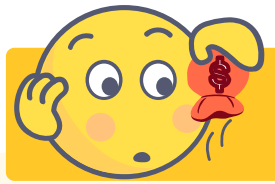
- 常规数据库：MySQL、PostgreSQL、Oracle、DB2、…
- Hadoop上的数据仓库：Hive、Spark SQL、…
- 新型分布式数据仓库/MPP：…
- 云数据仓库：Snowflake、…
- 数据库一体机：ExaData、…

其它数据分析与统计技术

- Python, Scala, Java, Kotlin, …

esProc SPL：低代码、高性能、轻量级、全功能

esProc SPL 有什么相对SQL的优势?

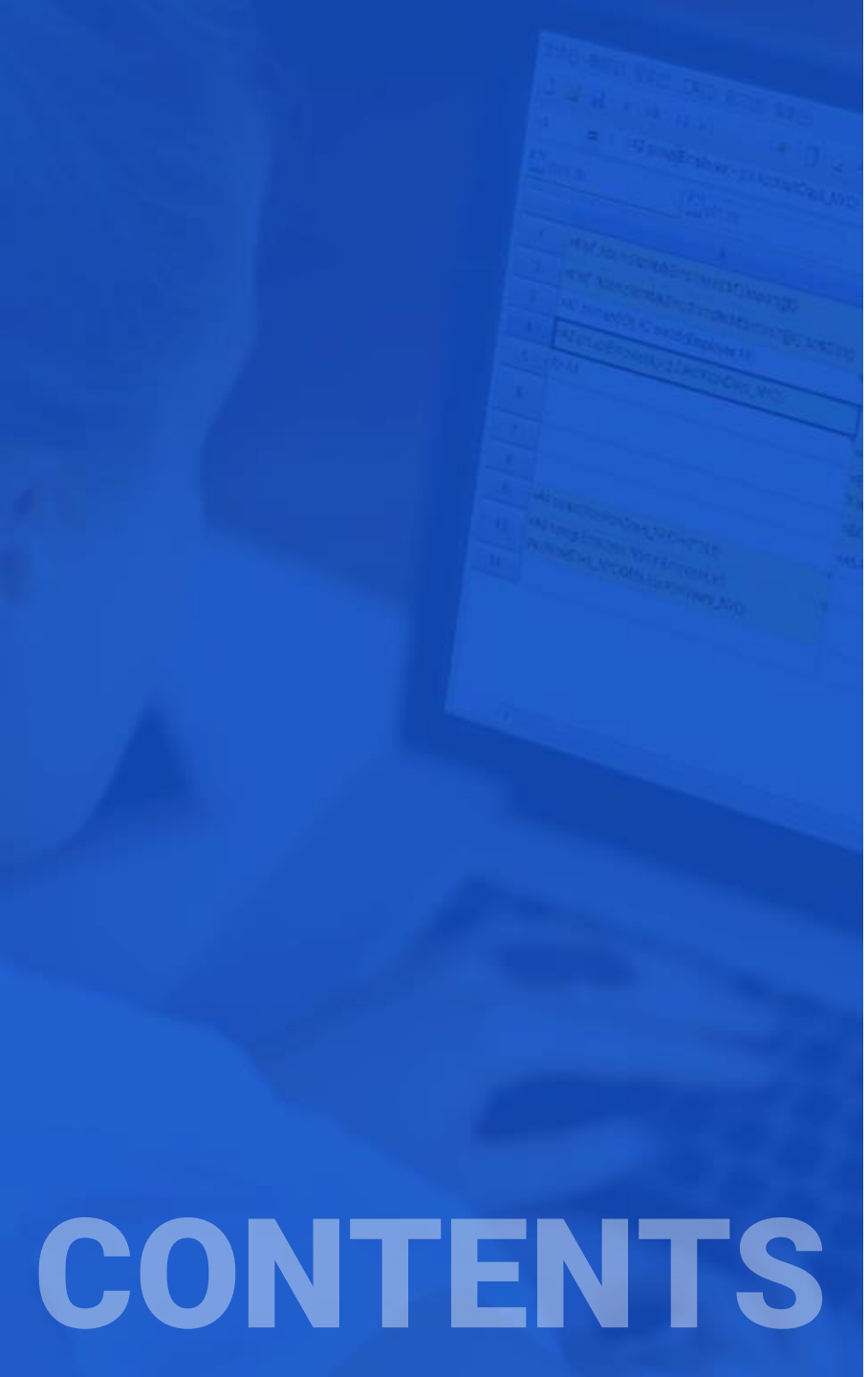


SQL

SPL

SQL		SPL
描述能力不足，复杂逻辑要用迂回写法		描述能力强大，自然思维实现复杂逻辑
冗长嵌套代码，难写难调试	开发成本	分步式代码，简洁易写易调试
计算量巨大消耗资源	硬件成本	低复杂度算法减少资源消耗
沉重封闭的计算能力导致臃肿架构	运维成本	可集成的开放计算能力获得轻盈架构
能力不完善迫使技术栈复杂化		独立完成绝大部分任务

esProc SPL：开发、硬件、运维成本全面降低X倍



CONTENTS

02

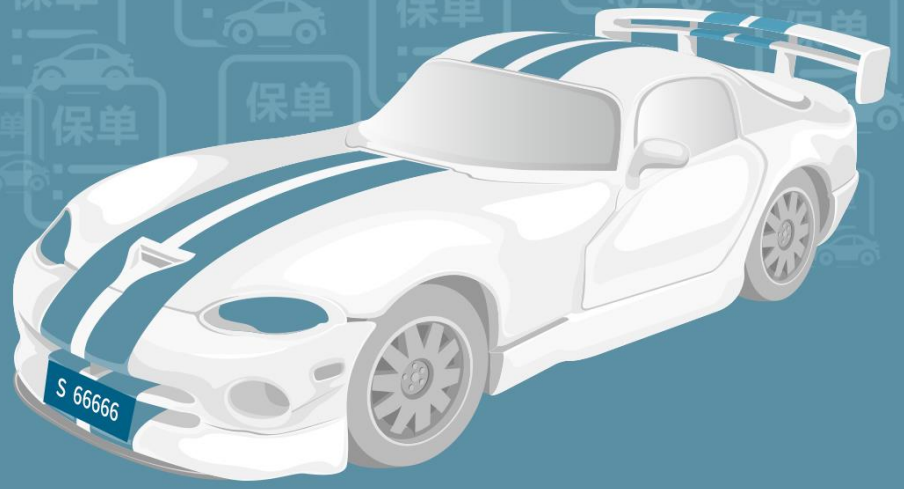
案例简析

案例 CASE

某保险公司车险跑批

问题与难点

- 保单表 3500万行，明细表 1.23亿行
- 关联方式多样需要分别处理



- Informix
- 10天新增保单关联 47分钟
- 30天新增保单关联 112分钟
- 代码 1800行

esProc SPL

- 10天新增保单关联 13分钟
- 30天新增保单关联 17分钟
- 代码 500行

提速 **6.5倍**

案例详情：http://c.raqsoft.com.cn/article/1594119021002

案例 CASE

某银行对公贷款业务跑批

问题与难点

- 48个SQL步骤，3300行
- 历史数据量1.1亿行，每日新增137万行
- 复杂多表关联

- 小型机AIX+DB2
- 运算时间 1.5小时

esProc SPL

- 用时 10分钟，代码 500行

提速 **8.5倍**

案例详情：http://c.raqsoft.com.cn/article/1596098387752

案例 CASE

手机银行多并发帐户查询

问题与难点

- 用户多，并发**访问量大**
- 机构信息经常变更，需要及时关联

- Hadoop上商用数仓无法满足高并发要求
- 换用6台ElasticSearch集群能应对并发，但不能实时关联，数据更新时间长，期间只能停止服务

esProc SPL

- 单机做到ES集群同样并发量
- 实时关联，机构信息更新零等待

1台顶6台

案例详情：http://c.raqsoft.com.cn/article/1595490353934

案例
CASE

某银行贷款去重户数指标统计

问题与难点

- 标签众多，数百个标签任意组合查询
- 2000万行大表及更大的明细表关联、过滤、汇总计算
- 每个页面涉及近200指标计算，10并发共2000多指标同时计算

- Oracle
- 无法实时计算，只能预先约定查询要求，提前一天预计算

esProc SPL

- 10并发共2000指标计算不到3秒
- 无需预先准备，临时选择任意标签组合，实时查询结果

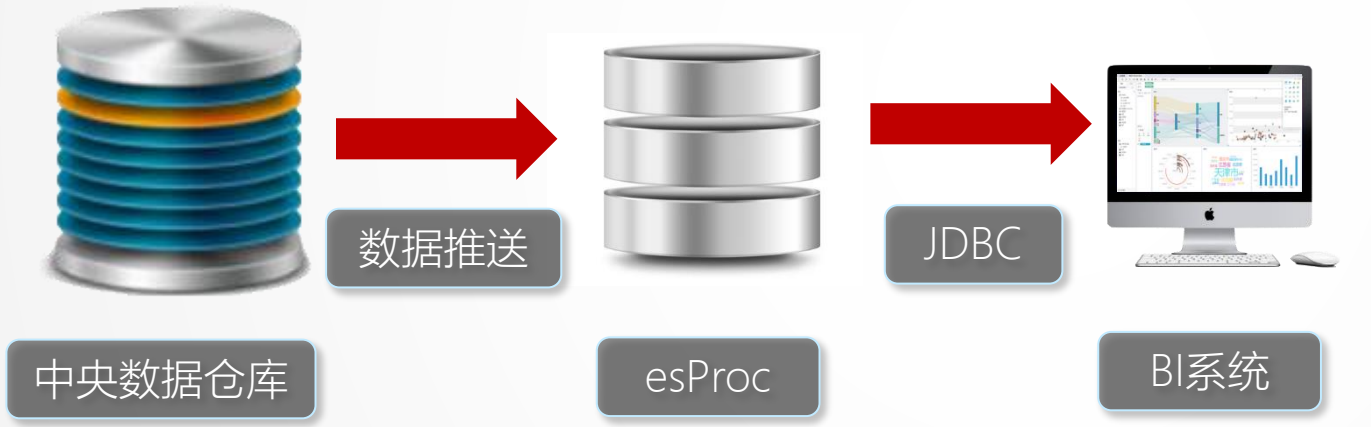
预计算变实时计算

案例详情：http://c.raqsoft.com.cn/article/1593424083742

某银行BI系统的前置数据库

中央数据仓库承担全行的数据任务，负担过重，只能分配给BI系统5个并发

仅对少量高频数据，DB2也无法胜任实时查询，更无法实现数据路由，需要用户选择数据源



5并发->100并发

esProc存储少量高频数据，大量低频数据仍存储在数据仓库中，避免重复建设

esProc承担绝大多数的高频计算任务，剩下少量低频任务自动路由到中央数据仓库

某保险公司-库外存储过程

Vertica不支持存储过程，要写异常复杂的嵌套SQL准备数据，经常还要借助Java代码

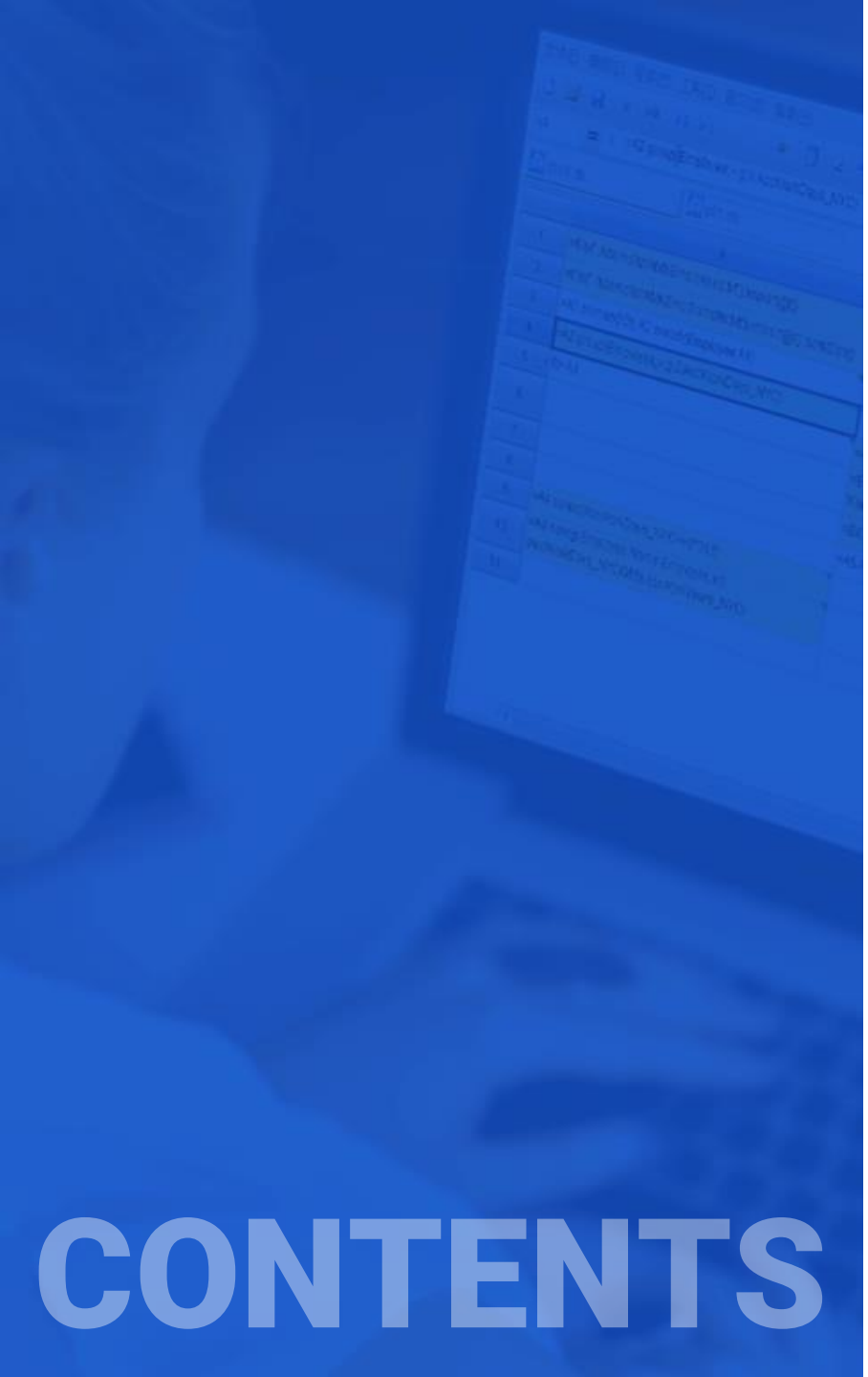
与MySQL的混合计算时要先将MySQL数据转入，繁琐、不实时、数据库臃肿

用户评价

The best use for us is to pass parameters to the Vertica database.

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.

引入esProc后，不仅能实现Vertica上类存储过程运算，还能直接跨源计算



CONTENTS

02

esProc凭什么?

SQL为什么难写：一支股票最长连涨了多少天

```
SELECT MAX(ContinuousDays)
FROM (SELECT COUNT(*) ContinuousDays
        FROM (SELECT SUM(UpDownTag) OVER ( ORDER BY TradeDate) NoRisingDays
                FROM (SELECT TradeDate,
                             CASE WHEN Price>LAG(price) OVER ( ORDER BY TradeDate)
                                  THEN 0 ELSE 1 END UpDownTag
                        FROM Stock ) )
        GROUP BY NoRisingDays )
```

SQL对有序运算支持不足，未直接提供有序分组，只能采用迂回思路，写成四层嵌套的形式

这样的句子不仅很难写出来，写出来想看懂也不容易

面对复杂的业务逻辑，SQL的复杂度会陡增，既难懂又难写

这并非罕见需求，现实中数千行的SQL代码中这种情况比比皆是，严重影响开发和维护效率

SQL为什么跑不快：1亿条数据取前10名

```
SELECT TOP 10 * FROM Orders ORDER BY Amount DESC
```

这个查询用了ORDER BY，严格按此逻辑执行，意味要将全量数据做排序，性能将很差

我们知道有不必全排序而完成这个运算的办法，但用SQL无法描述，只能寄希望于数据库的优化引擎

简单情况（比如本句），很多数据库都能优化，但情况再复杂一些，数据库优化引擎就会晕了

下面的分组内取前N名，SQL无法直接描述了，还是要采用迂回思路利用窗口函数写成子查询

面对这种迂回写法，数据库优化引擎也不会优化了，只能去执行排序

```
SELECT * FROM (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY Area ORDER BY Amount DESC) rn
    FROM Orders )
WHERE rn<=10
```

SPL的解决方法

	A
1	=Stock.sort(TradeDate).group@i(Price<Price[-1]).max(~.len())

这句SPL和前面SQL的运算逻辑相同，但SPL提供有序分组运算，描述起来直观简洁

	A	
1	=file(“Orders.ctx”).open().cursor()	
2	=A1.groups(;top(10;-Amount))	金额在前10名的订单
3	=A1.groups(Area;top(10;-Amount))	每个地区金额在前10名的订单

SPL将TopN视为返回集合的聚合运算，避免全排序；全集和分组时写法类似，不再迂回

SPL为什么更有优势

【类比】计算1+2+3+...+100=?

普通人这么算

1+2=3
3+3=6
6+4=10
10+5=15
15+6=21
21+7=28
...

高斯这么算

1+100=101
2+99=101
...

一共有50个101
50*101= 5050

聪明的高斯想到了便捷高效的算法，这里的关键是：使用了**乘法**！

SQL就象只有**加法**的算术体系，代码冗长，计算低效

SPL则相当于发明了**乘法**！简化书写，提高性能

延伸阅读： 写着简单跑得又快的数据库语言SPL

SQL的困难源于关系代数，理论问题无法用工程手段解决，虽然经过多年改善，面对复杂需求时依然困难重重

SPL基于完全不同的理论体系：离散数据集，提供更丰富的数据类型和基础运算，拥有更强大的表达能力

玩爆SQL的常见场景

1、复杂有序计算：用户行为转换漏斗分析

- 计算每个事件（页面浏览、搜索、加购物车、下单、付款等）后的用户流失率
- 多个事件在指定时间窗口内完成、按指定次序发生才有效，SQL难以实现，更难优化

2、多步骤大数据量跑批

- 复杂业务需求难以直接用SQL完成，游标读数计算慢，且难以并行，浪费计算资源
- 存储过程实现要几千行数十步，伴随中间结果反复落地，跑批时间窗口内完不成

3、大数据上多指标计算，反复用关联多

- 一次完成数百个指标的计算，多次使用明细数据，期间还涉及关联，SQL需要反复遍历
- 大表关联、条件过滤、分组汇总、去重计数混合运算，伴随高并发实时计算

现实业务中复杂SQL（及存储过程）动辄数百上千行，大量迂回思路才能完成运算，代码复杂、性能低下

电商漏斗运算

```
with e1 as (
   select uid,1 as step1,min(etime) as t1
   from event
   where etime>= to_date('2021-01-10') and etime<to_date('2021-01-25')
      and eventtype='eventtype1' and …
   group by 1),
e2 as (
   select uid,1 as step2,min(e1.t1) as t1,min(e2.etime) as t2
   from event as e2
   inner join e1 on e2.uid = e1.uid
   where e2.etime>= to_date('2021-01-10') and e2.etime<to_date('2021-01-25')
      and e2.etime > t1 and e2.etime < t1 + 7
      and eventtype='eventtype2' and …
   group by 1),
e3 as (
   select uid,1 as step3,min(e2.t1) as t1,min(e3.etime) as t3
   from event as e3
   inner join e2 on e3.uid = e2.uid
   where e3.etime>= to_date('2021-01-10') and e3.etime<to_date('2021-01-25')
      and e3.etime > t2 and e3.etime < t1 + 7
      and eventtype='eventtype3' and …
   group by 1)
select
   sum(step1) as step1,
   sum(step2) as step2,
   sum(step3) as step3
from e1
   left join e2 on e1.uid = e2.uid
   left join e3 on e2.uid = e3.uid
```

	A
1	=["etype1","etype2","etype3"]
2	=file("event.ctx").open()
3	=A2.cursor(id,etime,etype;etime>=date("2021-01-10") && etime<date("2021-01-25") && A1.contain(etype) && …)
4	=A3.group(uid).(~.sort(etime))
5	=A4.new(~.select@1(etype==A1(1)):first,~:all).select(first)
6	=A5.(A1.(t=if(#==1,t1=first.etime,if(t,all.select@1(etype==A1.~ && etime>t && etime<t1+7).etime, null))))
7	=A6.groups(;count(~(1)):STEP1,count(~(2)):STEP2,count(~(3)):STEP3)

SPL提供有序计算且集合化更彻底，直接按自然思维写出代码，简单且高效。
这段代码能够处理任意步骤数的漏斗，只要改变参数即可

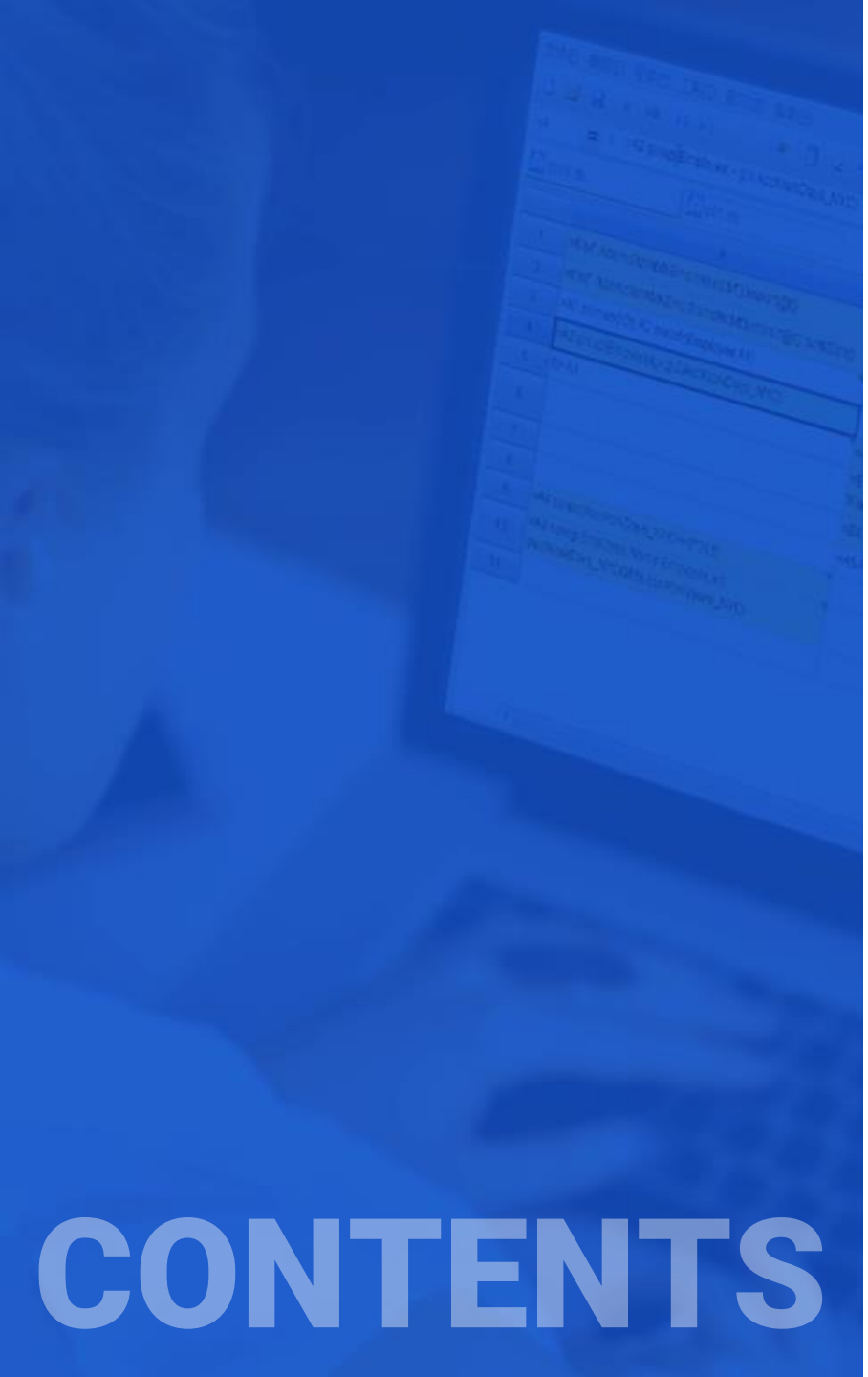
SQL缺乏有序计算且集合化不够彻底，需要迂回成多个子查询反复JOIN的写法，编写理解困难而且运算性能非常低下
限于篇幅，只写了三步漏斗，再增加步骤时还要增加子查询

SPL部分高性能计算机制

遍历技术	高效关联	高速存储	集群计算
延迟游标	※ 外键指针化	有序压缩存储	抢先式负载均衡
聚合理解	※ 外键序号化	列式存储	※ 集群复组表
※ 有序游标	有序归并	※ 层次序号式定位	※ 集群维表
※ 遍历复用	※ 附表	索引及缓存	※ 内存备胎式容错
预过滤遍历	※ 单边分堆连接	※ 倍增分段并行	外存冗余式容错

※ 这里许多算法和存储方案是SPL的独创发明！

延伸阅读： 结构化大数据高性能计算技术

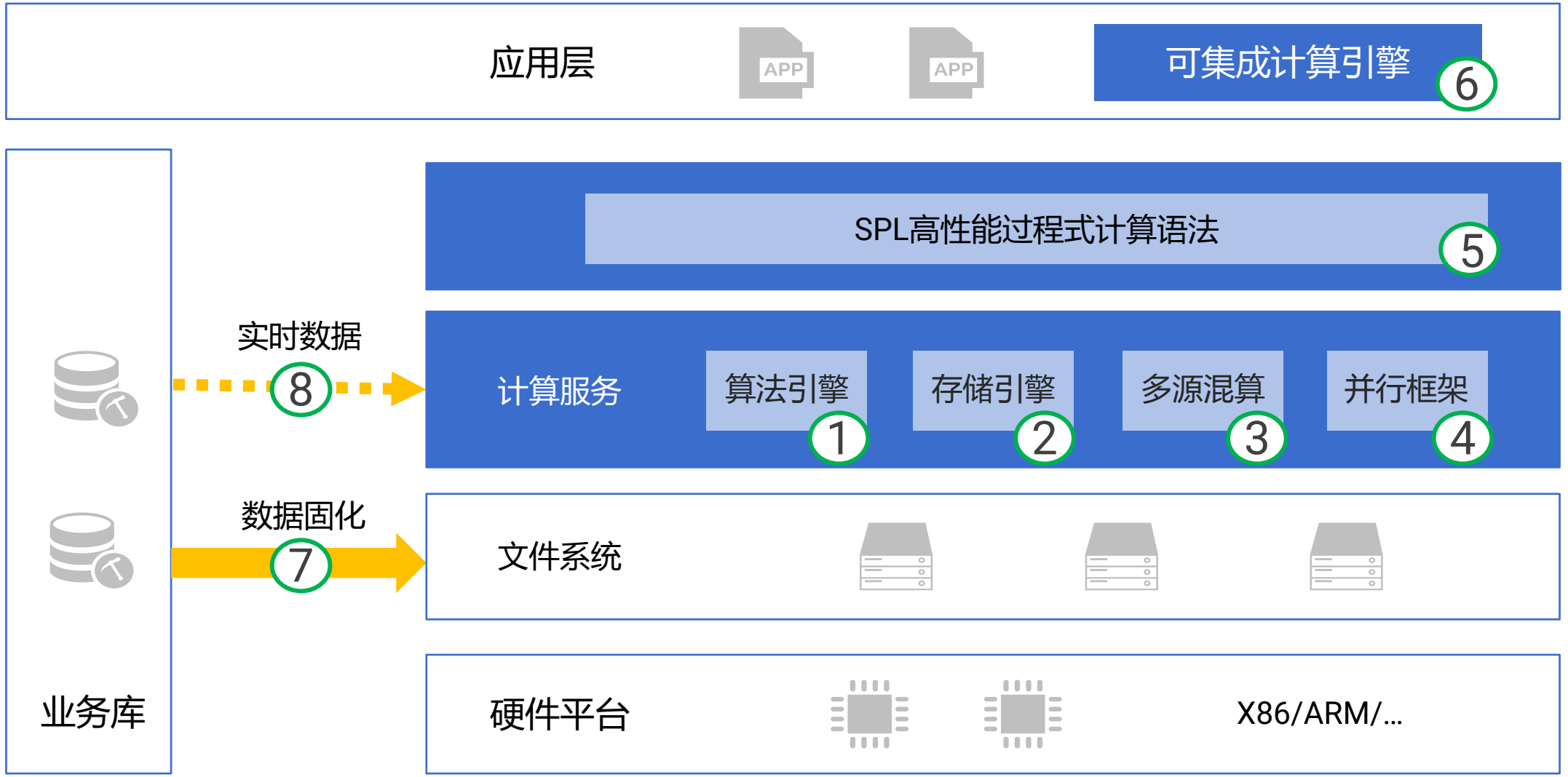


CONTENTS

04

技术特性

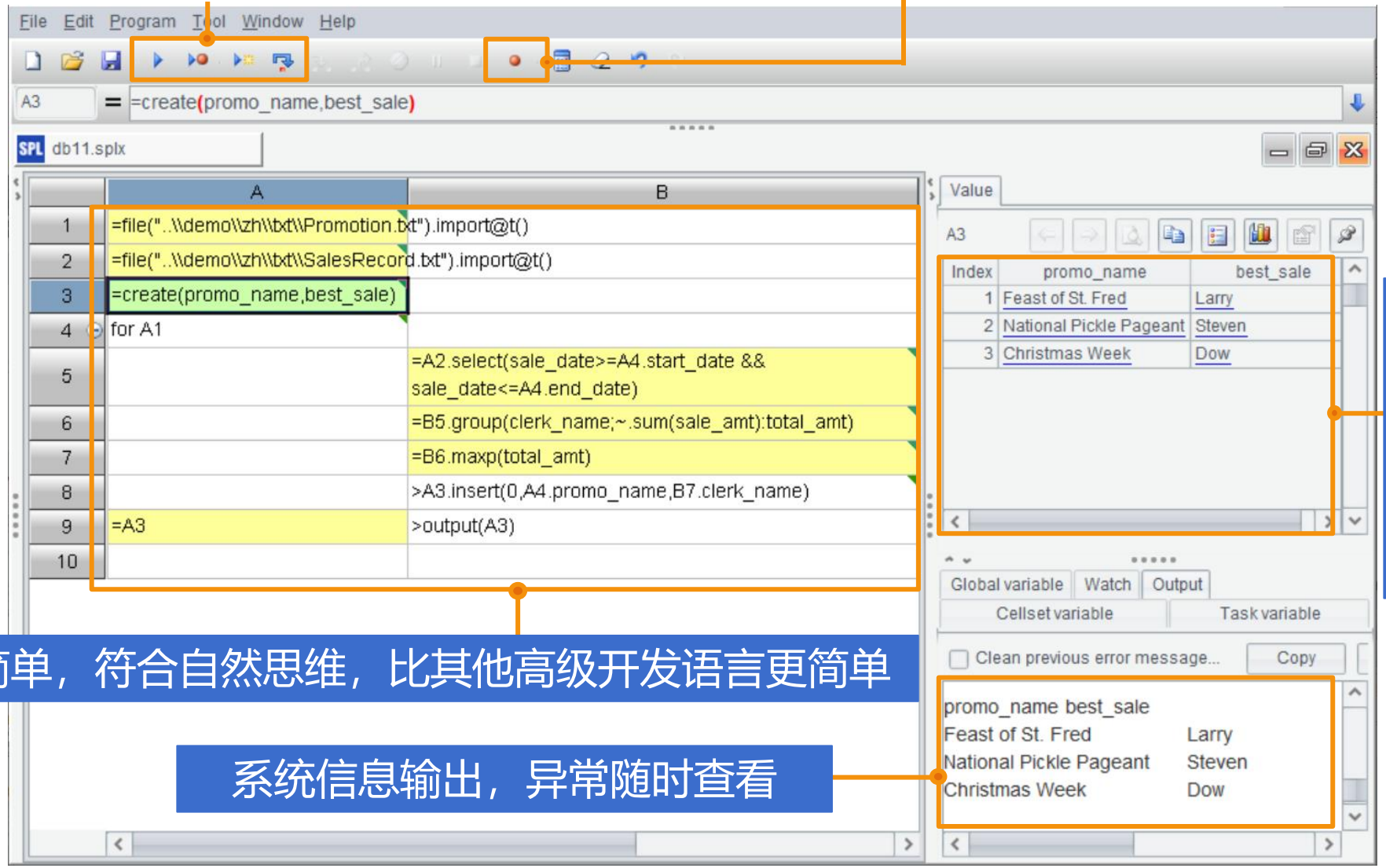
esProc 技术架构


应用层
APP
APP
可集成计算引擎
6
SPL高性能过程式计算语法
5
实时数据
8
计算服务
算法引擎
1
存储引擎
2
多源混算
3
并行框架
4
数据固化
7
文件系统
硬件平台
X86/ARM/...
业务库

简洁易用的开发环境

执行、调试执行、执行到光标、单步执行

设置断点



网格结果所见即所得，易于调试；方便引用中间结果

语法简单，符合自然思维，比其他高级开发语言更简单

系统信息输出，异常随时查看

专门设计的语法体系

SPL特别适合复杂过程运算

	A	B	C	D	E	F
1	=esProc.query("SELECT orderID as contract, orderDate as date, customer, amount, empID as salesman FROM sales where year(orderDate)=? OR year(orderDate)					
2	=esProc.query(select * from employeeInfo")					
3	>A1.run(salesman=A2.select@1(ID:A1.salesman))		/field value is record			
4	>A1.group(salesman)					
5	=create(salesman, thisyearAmount, lastyearAmount, growthRate, custNumber, bigCustNumber,bigCustProportion)					
6	for A4	=A6(1).salesman.name				
7		=A6.select(year(date)==year).sum(amount)				
8		=A6.select(year(date)==year-1).sum(amount)				
9		=B8/B7-1	/growth rate			
10		=A6.group(customer).(~.sum(amount))				
11		=B10.count()	/number of customer			
12		=B10.count(~>=10000)	/number of big customer			
13		=B12/B11				
14		=A5.insert(0,B6,B7,B8,B9,B11,B12,B13)				

天然分步、层次清晰、直接引用单元格名**无需定义变量**

丰富的运算类库

专门针对结构化数据表设计

	A	B	C
1	=esProc.query("SELECT orderID as contract,		/retrieve sales records
2	=A1.group(salesman)		
3	=create(salesman,thisyearAmount, lastyearAmount, custNumber, bigCustNumber)		
4	for A2	=A4(1).salesman	
5		=A4.select(year(date)==year).sum(amount)	
6		=A4.select(year(date)==year-1).sum(amount)	
7		=A4.group(customer).(~.sum(amount))	

分组、循环

	A	B	C
1	=file("traderecord.txt").import@t()		
2	=A1.sort(customerID, tradeDate)		
3	=A2.select(autoType=="Jetta" || autoType="Passat").dup@t()		
4	=A3.derive(interval(tradeDate[-1], tradeDate):space)		
5	=A4.select(autoType[-1]=="Jetta" &&atuoType=="Passat" &&customerID=customerI		
6	=A5.avg(space)		
7			

排序、过滤

	A	B	C
1	=esProc.query("select * from employee")		
2	=A1.select(sex=="male")		
3	=A1.select(birthday>=date("1970-01-01"))		
4	=A2^A3	/intersect, find out male employee born after 1970	
5	=A2&A3	/union, find out male employee or employee born after 1	
6	=A2\A3	/subtract, find out male employee born before 1970	
7	=A4.sum(salary)		
8	=A5.avg(age)		
9	=A5.sort(birthday)		

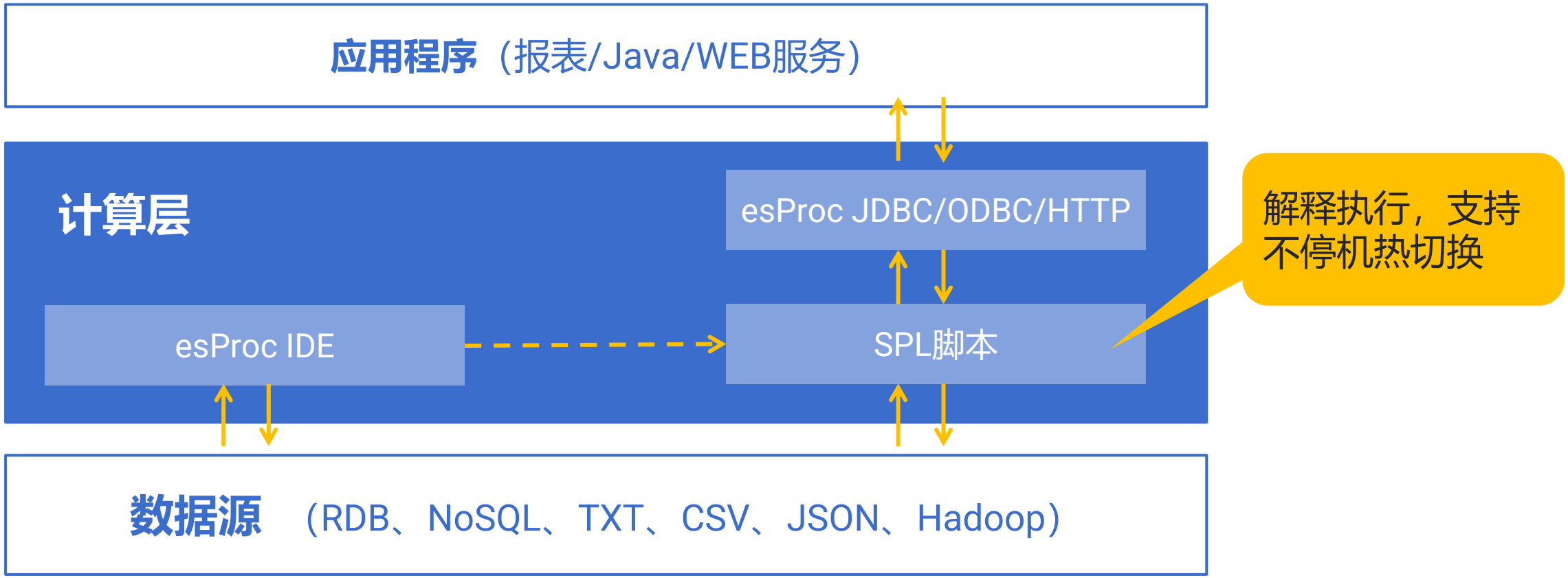
集合运算

	A	B	C
1	=esProc.query("select * from employee")		
2	=A1.sort(entryDate)		
3	=A2.pmin(birthday)	/select recordNo of employee born at earl	
4	=A2(to(A3-1))	/directly access employee record via reco	
5	=esProc.query("select * from stock where	stockCode="000062")	
6	=A5.sort(tradeDate)		
7	=A6.pmax(closePrice)	/recordNo of highest exchange closing qu	
8	=A6.calc(A7,closePrice/closePrice[-1]-1)		

有序集合

超强集成性

esProc采用Java开发，可独立运行，也可无缝集成到应用



多样性数据源支持

直接使用多个数据源混合计算

无需导入数据库即可计算，保持原有数据源优势，更好的实时性

灵活高效的文件存储

高性能

私有数据存储格式，集文件、组表

文件系统

可用树状目录方式按业务分类存储数据

集文件

倍增分段方式支持任意数量并行

自有高效压缩编码（减少空间；CPU占用少；安全）

泛型存储，允许集合数据

组表

行列混合存储

有序存储提高压缩率和定位性能

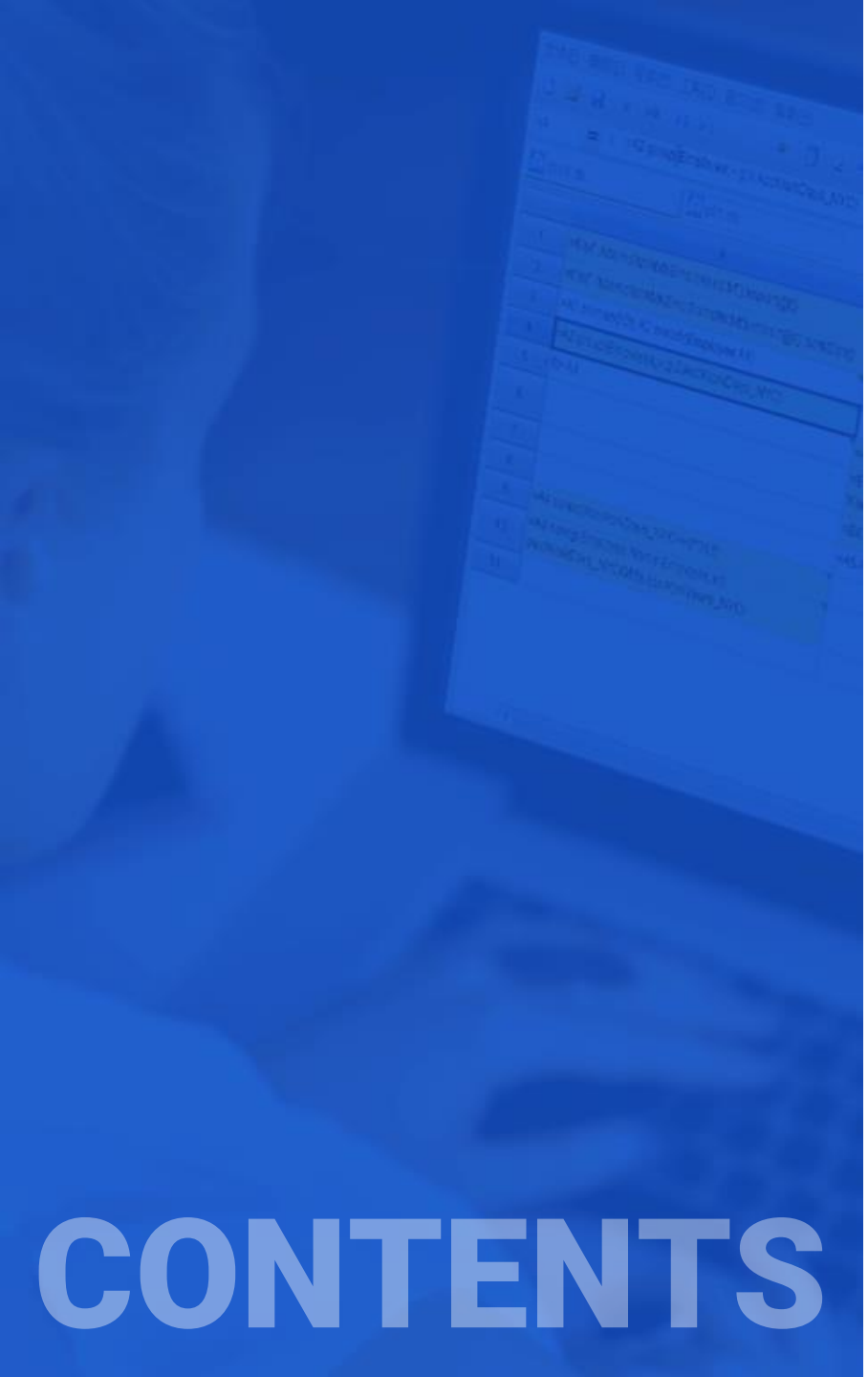
高效**智能索引**

倍增分段方式支持任意数量并行

主子表合一减少存储与关联

排号键值实现高效定位关联

直接文件存储，无需数据库，更高效更灵活，也更便宜



CONTENTS

05

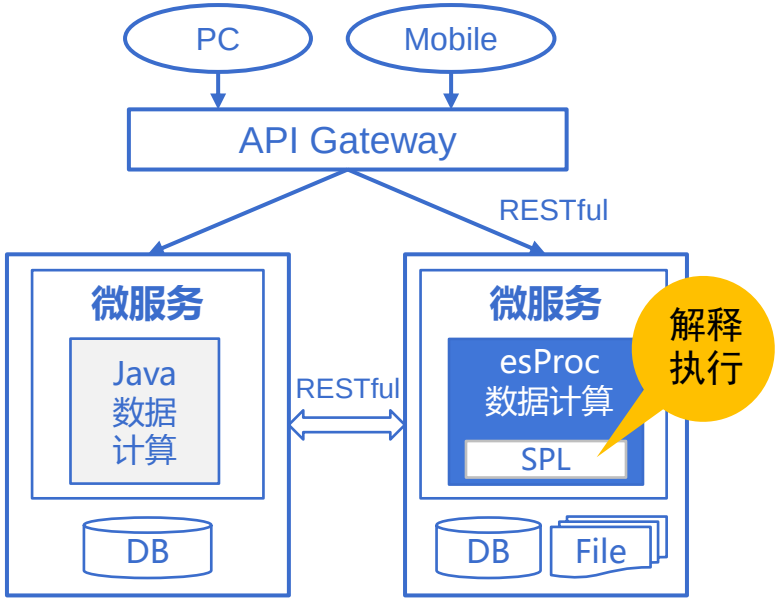
更多方案

数据型微服务实现

问题

- 微服务等主流架构要求在应用端实施数据处理
- 数据库很难嵌入前端应用使用，只能硬编码
- Java/ORM缺乏结构化计算类库导致数据处理开发困难，无法热切换

解决方案



- ✓ SPL替代Java/ORM在（微服务）应用内实施数据计算
- ✓ 丰富计算类库与敏捷语法简化开发
- ✓ 体系开放可以实时处理任意数据源数据
- ✓ SPL解释执行，天然支持热切换
- ✓ 高效算法与并行机制保证计算性能

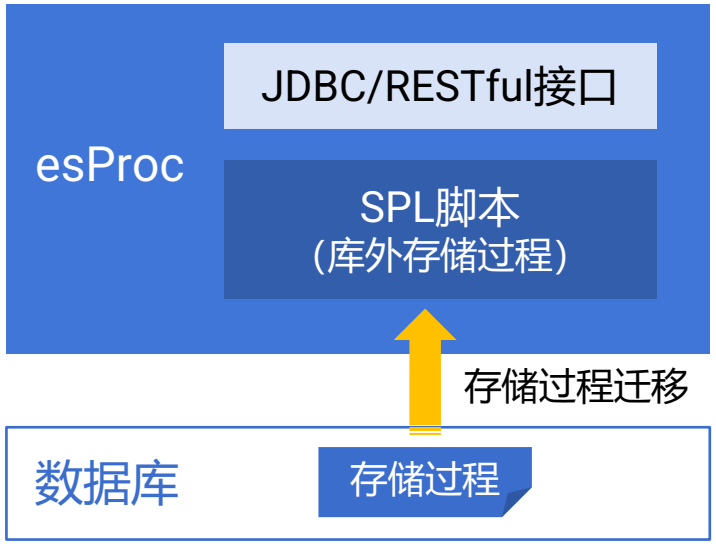
延伸阅读：开源 SPL，ORM 的终结者

替代存储过程

问题

- 存储过程编辑调试困难，缺乏移植性，
- 编译存储过程对权限要求过高，安全性差
- 存储过程被多个应用共用还会造成应用间紧耦合

解决方案



- ✓ SPL天然适合复杂多步数据计算
- ✓ SPL脚本天然可移植
- ✓ 脚本只要求数据库的读权限，不会造成数据库安全问题
- ✓ 不同应用的脚本分别存储于不同目录，不会造成应用间耦合

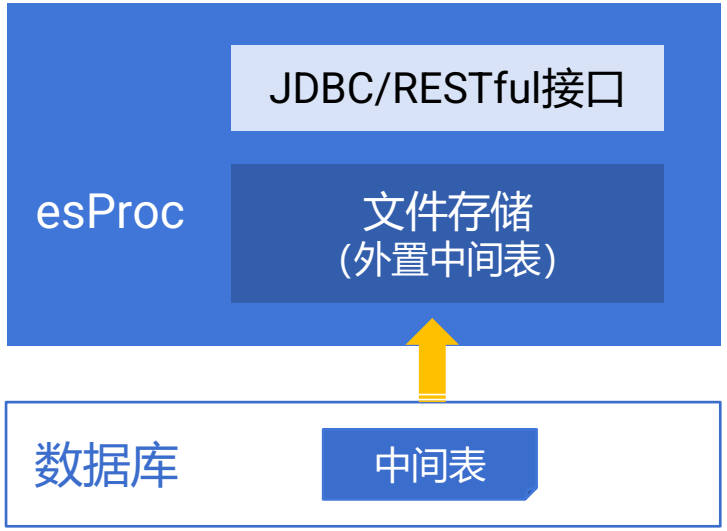
延伸阅读：爱恨交加的存储过程该往何处去

消灭数据库中间表

问题

- 为了查询效率或简化开发会在数据库中生成大量中间表
- 中间表空间占用大，导致数据库过于冗余臃肿
- 不同应用使用同一个中间表会造成紧耦合，中间表管理困难（很难删除）

解决方案



- ✓ 中间表存储在数据库中是为了利用数据库进行后续计算，使用文件存储后可使用SPL实施后续计算
- ✓ 外置中间表（文件）更易于管理，采用不同目录存储不会引起应用耦合性问题
- ✓ 中间表外置可以为数据库充分减负，甚至不需要部署数据库

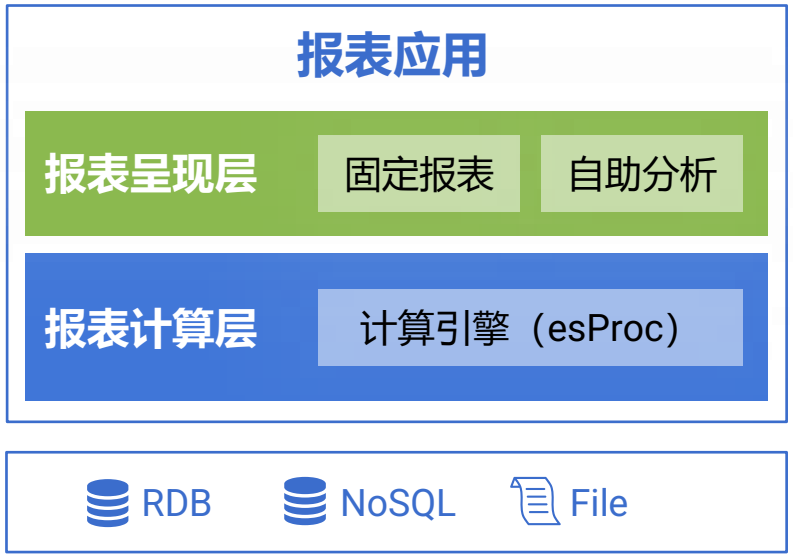
延伸阅读：开源 SPL 消灭数以万计的数据库中间表

应对报表没完没了

问题

- 报表/BI工具只能解决呈现环节的问题，对数据准备无能为力
- 数据准备使用SQL/存储过程/Java硬编码，开发维护困难，成本高
- 报表需求客观上没完没了，数据准备是导致高开发成本的主要因素

解决方案



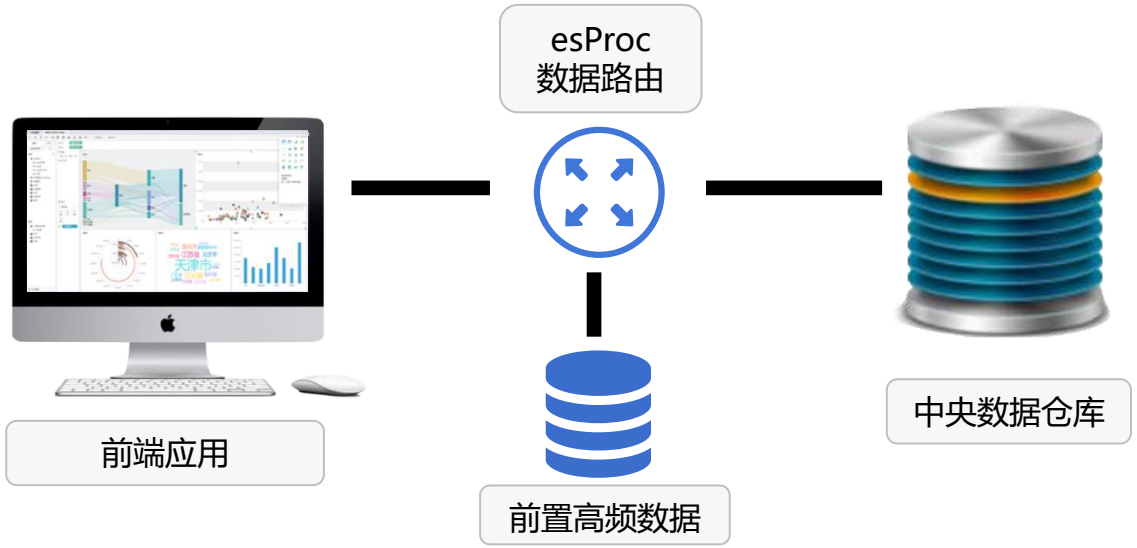
- ✓ 在报表呈现和数据源之间增加计算层来解决数据准备问题
- ✓ SPL简化报表数据准备，弥补报表工具计算能力不足，全面提升报表开发效率
- ✓ 报表呈现和数据准备两个环节都能快速响应以低成本应对没完没了的报表需求

延伸阅读：开源 SPL 优化报表应用应对没完没了

可编程路由实现前置计算

数据仓库任务繁重，需要前置计算分担压力

- 仅高频数据前置，接管大部分计算请求
- 可编程路由能力自动选择前置库和数据仓库并混合计算结果，应用可透明访问全量数据



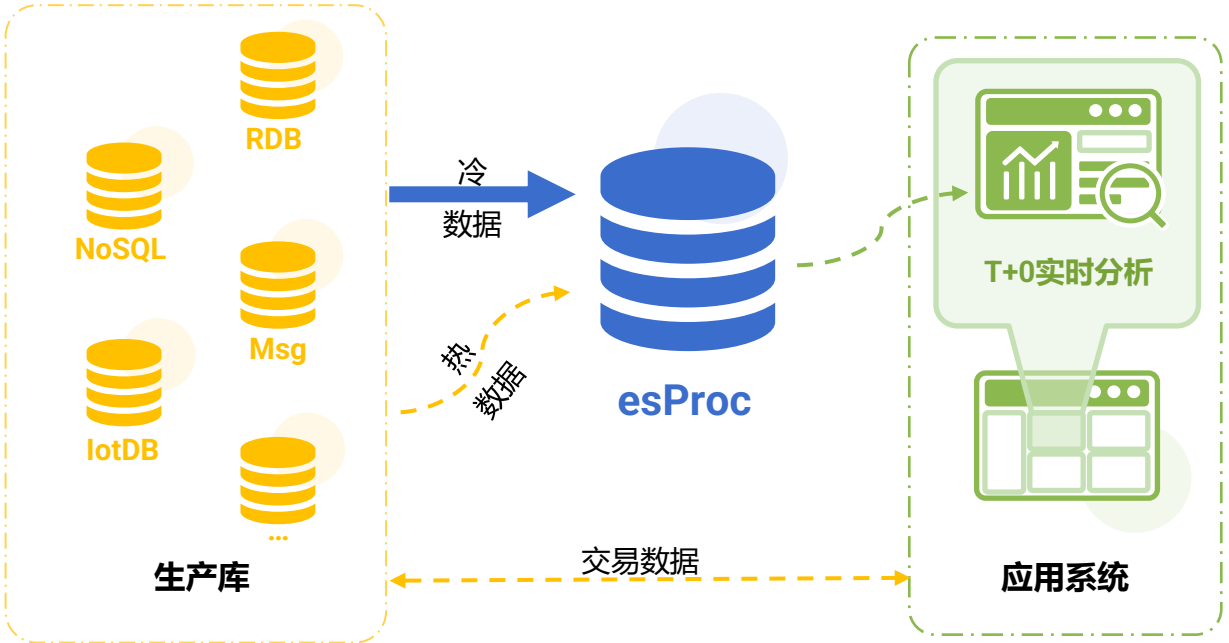
路由能力实现低成本高性能全量数据分析

- 传统数据库实现前置计算的尴尬
  - 前端应用可能访问所有数据，全量数据前置造成重复建设，成本高
  - 缺乏路由能力，仅部分数据前置时，应用不可透明访问全量数据，体验差

延伸阅读： http://c.raqsoft.com.cn/article/1670227164218

混合计算实现实时HTAP

- 冷热混算实现T+0实时分析
  - 历史冷数据精心整理
  - 交易热数据实时读取
- 生产系统无须改造
  - 充分利用原有多样数据源优势



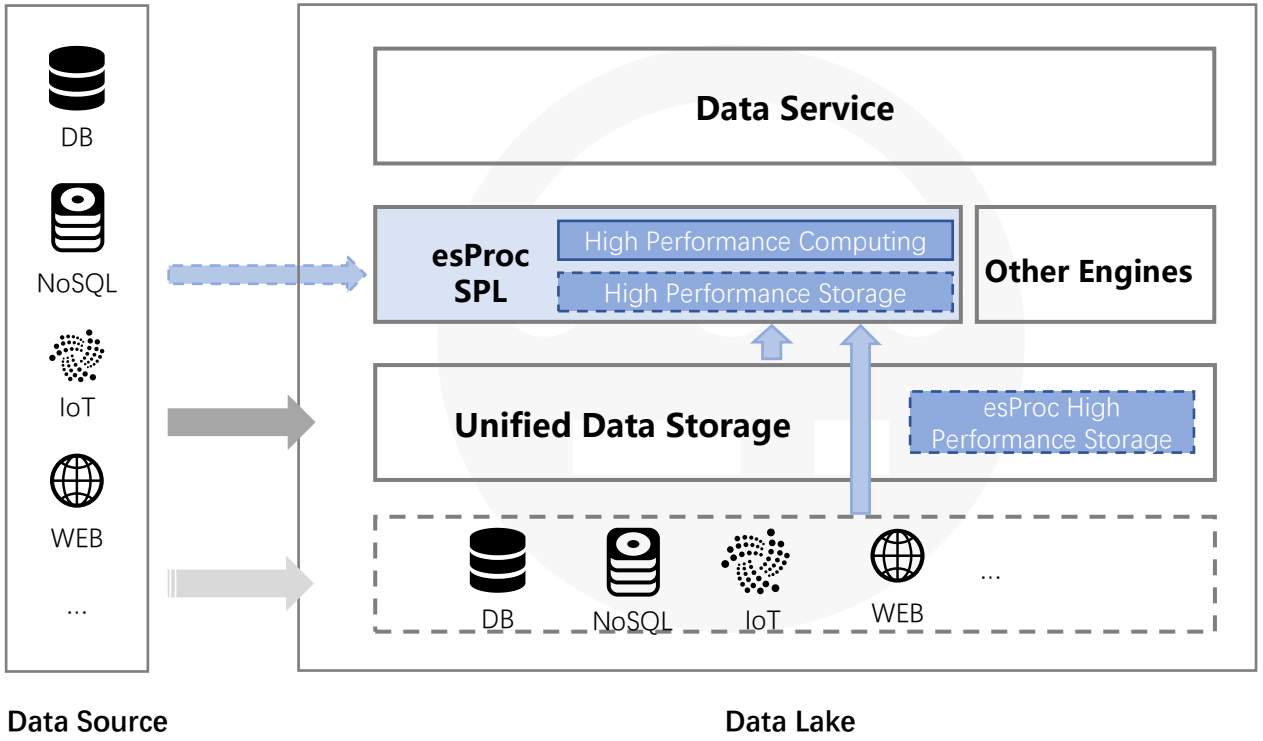
用开放的多源混算能力支撑低风险高性能强实时HTAP

- HTAP数据库难以实现HTAP需求
  - 要求更换生产系统数据库，风险大
  - SQL算力不足，历史数据准备不充分，性能低
  - 计算能力封闭，外部多样性数据源要求复杂ETL过程，实时差

延伸阅读： http://c.raqsoft.com.cn/article/1658456594393

文件计算实现湖仓一体

- 公开格式文件数据计算
  - txt/csv/xls/json/xml
- 高性能私有格式文件存储与计算
- 随意进入，逐步整理；Lake to House
- 丰富数据源接口，直接实时计算



- RDB只能House不能Lake
- 强约束，不合规的数据进不来，复杂ETL过程低效
- 封闭性，外部数据无法计算，更不能混合实时计算

延伸阅读： http://c.raqsoft.com.cn/article/1659057410089

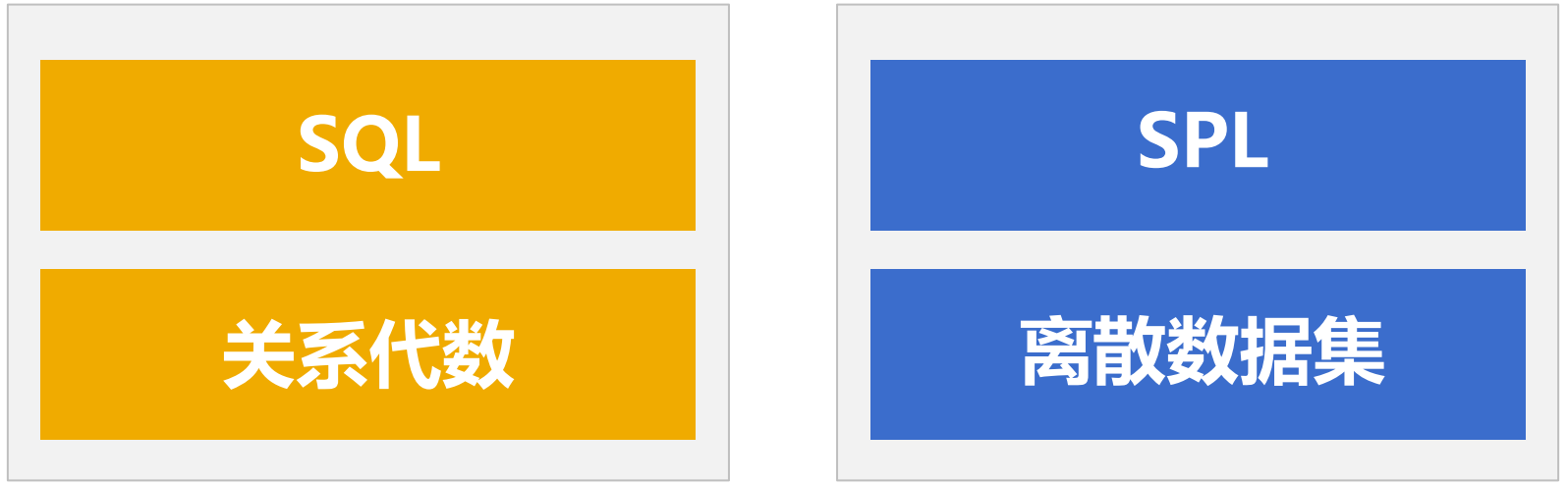
FAQ

常见问题

esProc基于开源或数据库技术吗?

A

esProc基于全新的计算模型，无开源技术可以引用，从理论到代码全部自主创新



基于创新理论的esProc不能再使用SQL实现高性能，SQL无法描述大部分低复杂度算法

Q esProc可以部署在哪里?

A

esProc 完成用纯Java开发，可以在任何有JVM的操作系统下运算，包括虚拟机、云服务器以及容器。

Q 应用程序如何调用esProc?

A

esProc 提供了标准的JDBC驱动，可被Java应用无缝集成调用。

对于非Java应用，esProc提供了HTTP/RESTFul调用接口。

Q esProc能基于现有数据库工作吗?

A

当然没问题!

不过，因为数据库I/O性能不佳， esProc在这种情况下不能保证高性能， 而且数据库通常也不提供低复杂度算法需要的存储。要获得高性能，则需要esProc自己存储数据

Q esProc把数据存在哪里?

A

esProc使用私有格式的文件来存储数据，任何操作系统下的文件系统都支持，包括网络文件系统。这样，esProc可以天然地实现存算分离机制。

esProc有什么不足吗?

和RDB相比

esProc的元数据能力相对不成熟，大部分计算要从访问数据源（文件）开始，对于简单运算会有些麻烦。

和Hadoop/MPP相比

esProc提供有集群能力，但缺乏机会历练。

历史上，esProc多次用单机替代原有的集群，仍获得同样甚至更高的性能。

和Python相比:

SPL 正在发展人工智能功能，但目前和Python相差还比较远。

SPL有多难学

SPL专门用于低代码和高性能

学习SPL不难，数小时即可掌握，数周就能熟练

难的是设计优化算法!

有没有将SQL自动转成 SPL的工具?

目前还没有。

SQL语句的信息不足， 需要猜出其目标才能设计合理的优化路径。我们优化SQL的经验远不如传统数据库厂商，这时候直接把SQL转换成SPL通常会导致更差的性能。

Q 如何启动性能优化项目?

A

最初的2-3个场景，由我方工程师介入配合用户实现

大多数程序员习惯了SQL思维方式，不熟悉高性能算法，需要用一两个场景训练和理解

几十种性能优化套路经历过也就学会了，算法设计和实现并不是那么难

授人以鱼不如授人以渔!

延伸阅读： SQL(及存储过程)跑得太慢怎么办

esProc优势总结

性能卓越

大数据处理速度比传统方案
平均提高1个数量级

高效开发

语法过程化，符合自然思维
丰富类库

5大优势

灵活开放

多源混合计算
独立、嵌入多种应用方式

资源节省

单机顶集群，减少硬件开销
绿色环保

成本锐减

开发、硬件、运维成本降低
X倍


THANKS