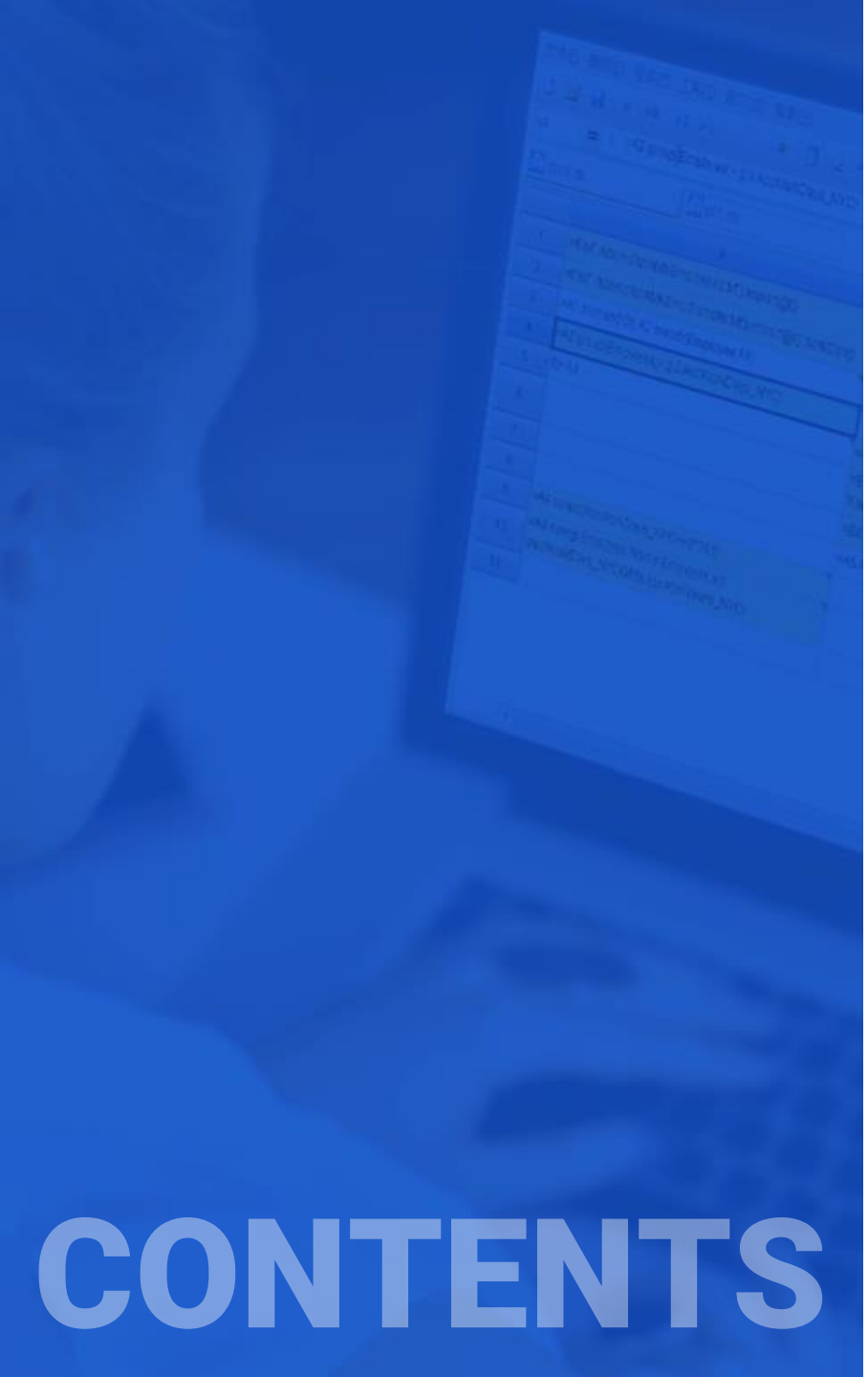
# esProc SPL

## Data analysis engine

**Low code / High performance / Lightweight / Versatility**



CONTENTS

01

# What is esProc?

# What is esProc SPL?

- Data computing and processing engine, run as an analysis **database** or **middleware.**
- Computing and processing of structured and semi-structured data
- Offline batch job, online query
- neither **SQL system** nor NoSQL Technology
- **Self created SPL syntax**, more concise and efficient


**SPL: Structured Process Language**

# What pain points does esProc SPL solve?

For the data computing scenarios : **Offline Batch Job** , **Online Query/Report**

**Slow**

- ✓ Slow batch jobs can not fit in certain time window, being strained especially on critical summary dates
- ✓ Being forced to wait for minutes for a query/report, the business personnel becomes angry
- ✓ More concurrencies, longer the query time span, the database crashes
- ✓ ……

**Difficult**

- ✓ N-layer nested SQL or stored procedures of dozens of KBs, programmer himself is confused after three months
- ✓ Dozens of datasources like RDB/NoSQL/File/json/Web/..., cross-source mixed computation is highly needed
- ✓ Separate the hot data and cold data into different databases, it is hard to perform real-time queries on the whole data
- ✓ ……

**Complex**

- ✓ Too much relied on the stored procedures, the application can not migrate, framework is hard to adjust
- ✓ Too many intermediate tables in the database, exhausting the storage and resources, but dare not to delete them
- ✓ Endless report demands in an enterprise, and how can the cost of personnel be relieved?
- ✓ ……

# What are the counterpart technologies of esProc SPL?

## Databases that use **SQL** syntax and are applied to **OLAP** scenarios

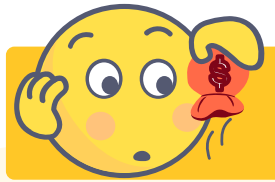
- Common database: MySQL, PostgreSQL, Oracle, DB2, …
- Data warehouse on Hadoop:  Hive, Spark SQL, …
- Distributed data warehouse/MPP: …
- Cloud data warehouse: Snowflake, …
- All-in-one database machine: ExaData, …

## Other technologies for structured data analysis and statistics

- Python, Scala, Java, Kotlin, …

**esProc SPL: Low code, High performance, Lightweight, Openness**

# What does esProc SPL bring beyond SQL?

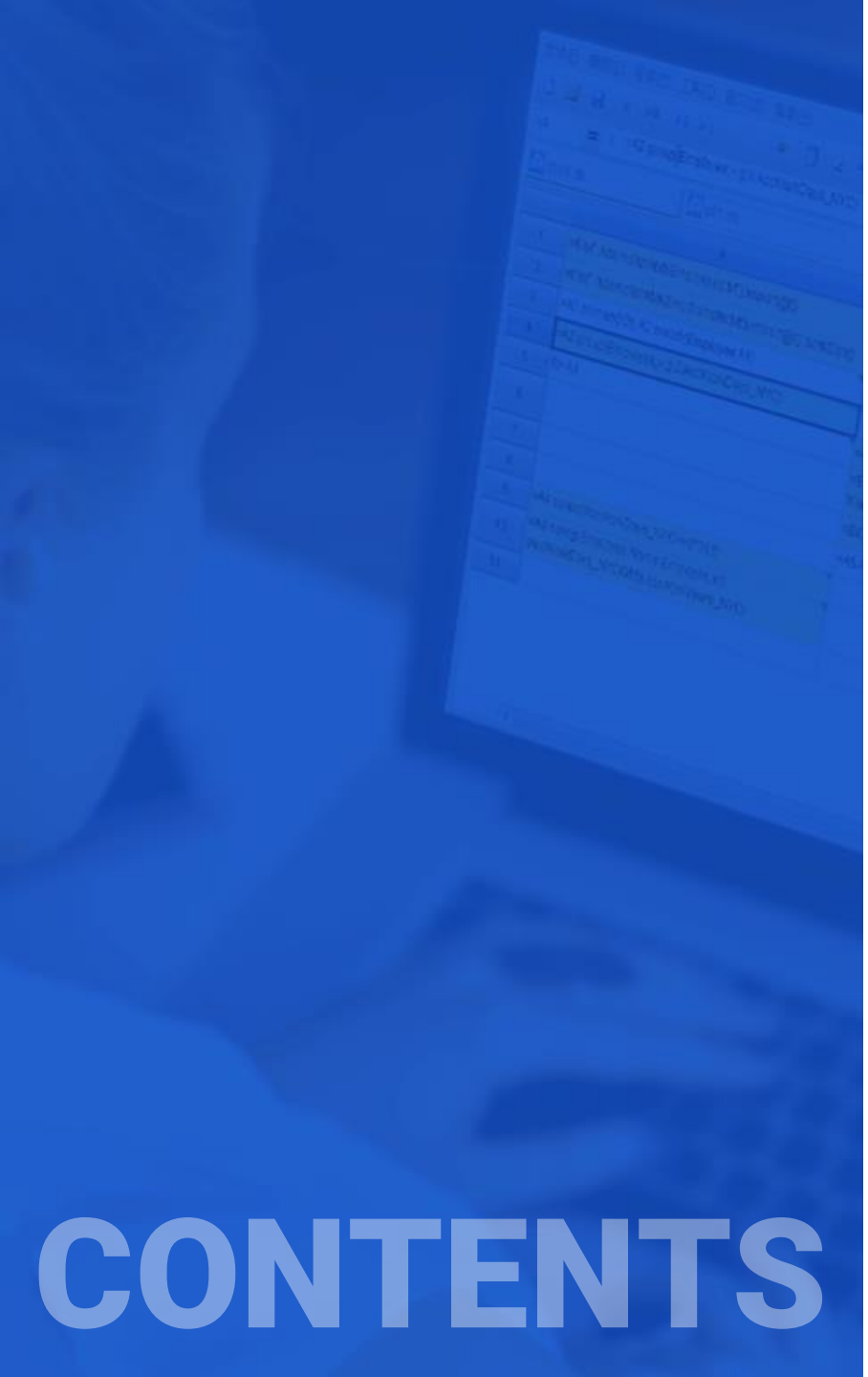


## SQL

## SPL

| SQL | | SPL |
|---|---|---|
| **The description ability is insufficient, and complex logic needs to be written in a circuitous way** | | **Strong description ability, natural thinking to implement complex logic** |
| Lengthy nested code, difficult to write and debug | Development cost | Simplified stepwise code, easy to write and debug |
| Huge computing loads consume resources | Hardware cost | Low complexity algorithms reduce resource consumption |
| Heavy and closed computing ability leads to bloated framework<br>Imperfect capabilities force complex technology stack | O&M cost | Integrated and open computing ability forms lightweight framework<br>Versatility enables implementing most tasks on its own |

**esProc SPL: Reduce the development, hardware, operation and maintenance costs by X times**



CONTENTS

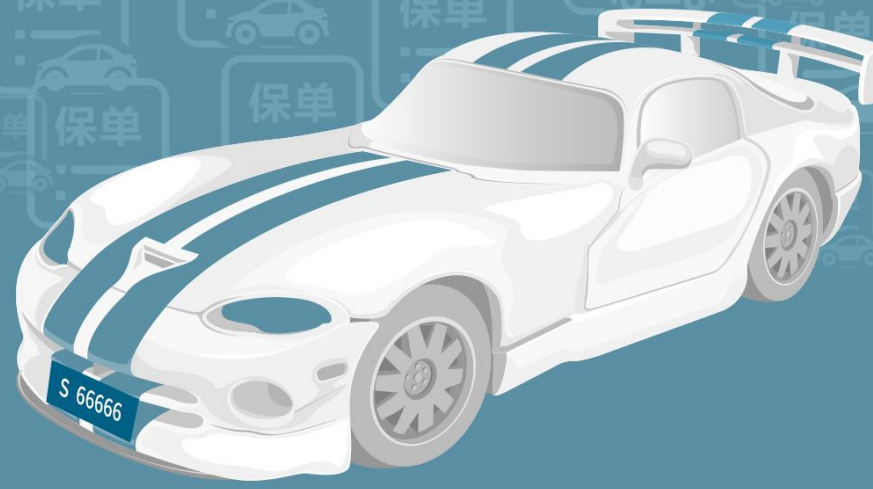
02

## Case Brief

# Case Batch job of insurance policies of an auto insurance company

**TASK**

- Insurance policy table: 35 million rows， details table: 123 million rows
- There are various ways of association, which need to be handled separately




- Informix
- Historical policy matching 6672 seconds
- Codes 1800 lines

esProc SPL

- Historical policy matching 1020 seconds
- Codes 500 lines

Increase speed

**6.5 times**

Case details： http://c.raqsoft.com/article/1644827119694

# Case Batch job of loan agreements of a bank

**TASK**

- SQL: 48 steps，3300 lines
- Historical data: 110 million rows，daily increase: 1.37 million rows
- Complex multi-table join

- AIX+DB2
- Calculation time: 1.5 hours

esProc SPL

- Calculation time: 10 minutes，codes: 500 lines

Increase speed

**8.5 times**

Case details：http://c.raqsoft.com/article/1644215913288

# Case Mobile banking: multi concurrent account query

**TASK**

- Huge number of users, and large concurrent accesses
- Branch information changes frequently and needs to be associated in time

- Commercial data warehouses on Hadoop cannot meet the high concurrency requirements
- Using 6 ElasticSearch cluster can cope with concurrency, but can not associate in real time. The data update time is long, and the service must be stopped during this period.

## esProc SPL

- Single machine can cope with the same concurrent volume as ES cluster
- Real time association, zero waiting time for branch information update

**1 server VS 6 servers**

Case details: *http://c.raqsoft.com/article/1643533607375*

# Case Calculation of the number of unique loan clients of a bank

**TASK**

- Too many labels, and hundreds of labels can be arbitrarily combined to query
- Association, filtering and aggregation calculation of a 20 million rows large table and even larger detailed tables
- Each page involves the calculation of nearly 200 indexes, and 10 concurrency will cause the concurrent calculation of more than 2000 indexes

- Oracle
- Unable to calculate in real time; The query requirements have to be submitted in advance, and the calculation is carried out one day earlier.

esProc SPL

- 10 concurrency, 2000 indexes in total, less than 3 seconds
- No need to prepare in advance, instantly select any label combination, and get query results in real time

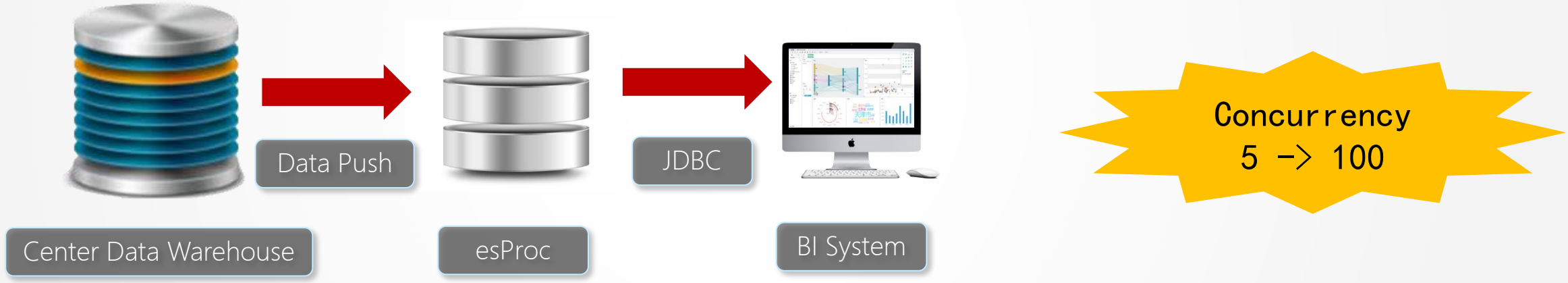
**Turn pre-calculation into real-time calculation**

Case details: http://c.raqsoft.com/article/1672279362788

# Front-end database in BI System of a bank

Center data warehouse undertakes all data task of whole bank, which is overburdened and can only assign 5 concurrencies to BI system

Only for a small amount of high-frequency data, DB2 is not capable for real-time query, and also unable to achieve data routing, users must select the data source



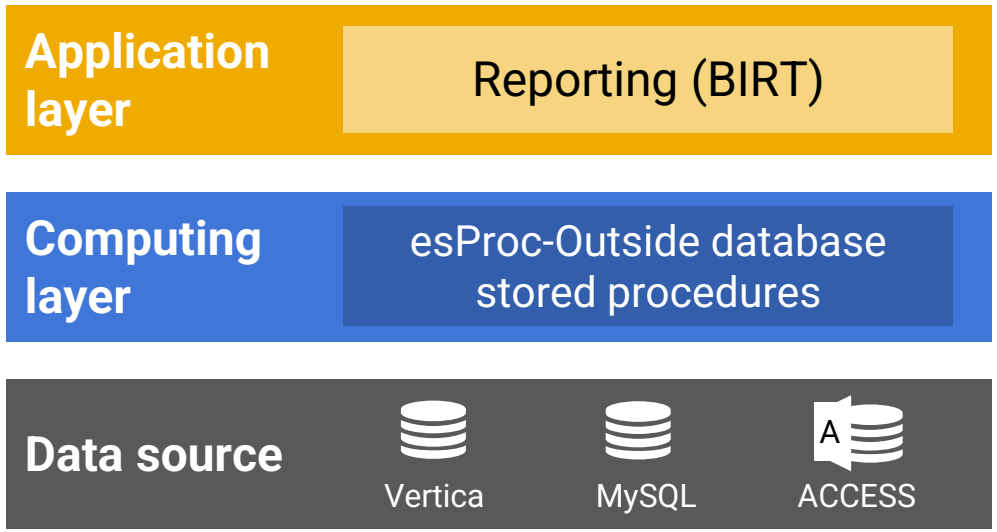

esProc stores a small amount of high-frequency data, and large low-frequency data is still stored in the data warehouse to avoid repeated construction

esProc takes over the most high frequency computing tasks, and a few low frequency tasks are automatically routed to the center data warehouse

# An insurance company-Outside database stored procedure

Vertica does not support stored procedures; To prepare data, complex nested SQL statements have to be written, and Java hardcoding is often required.

When mixed computing with MySQL, MySQL data has to be loaded into Vertica first, which is tedious, not real-time, and the database is bloated.

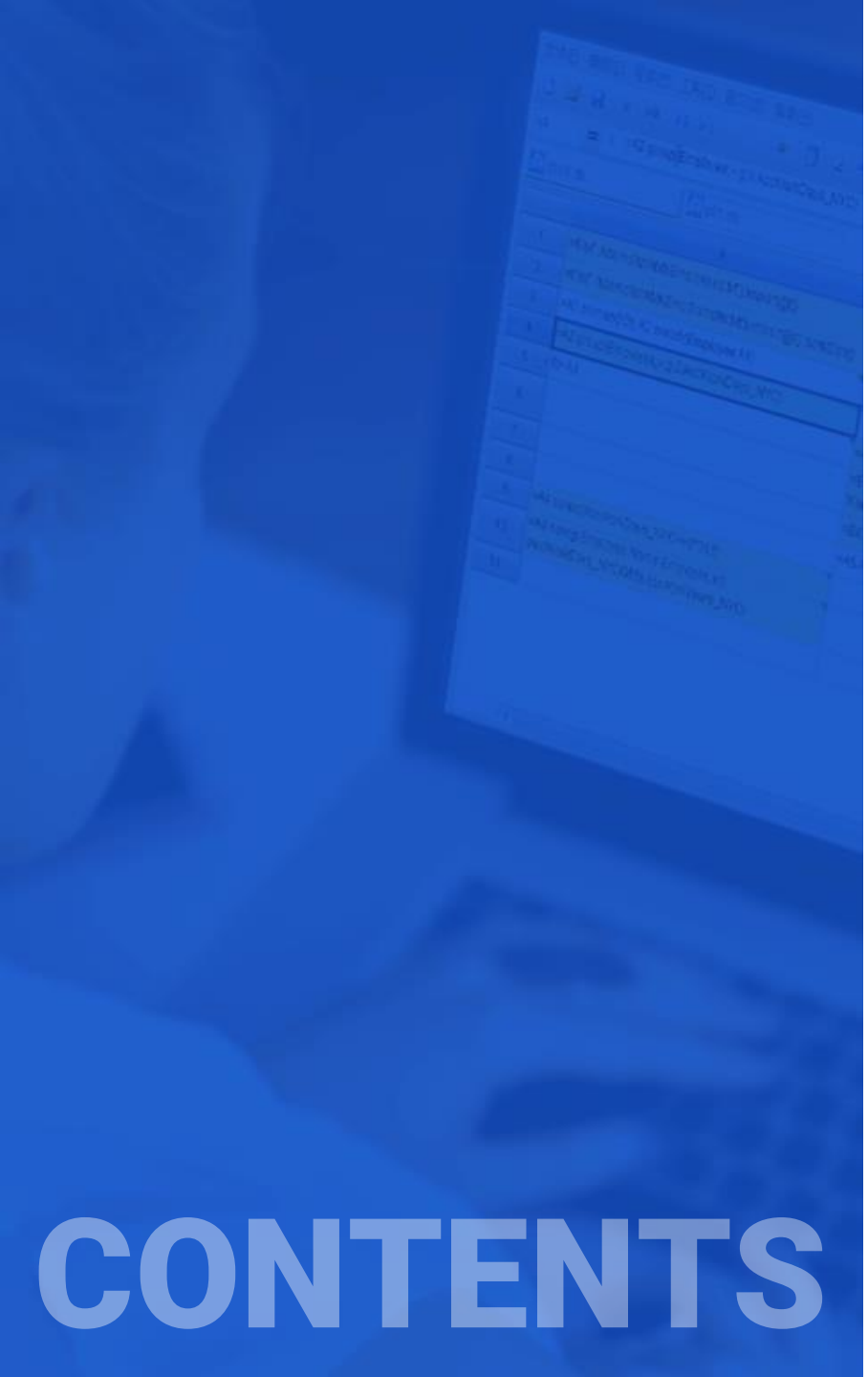



## User comments

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.

esProc can not only implement stored-procedure-like computations on Vertica, but also calculate different sources directly.



CONTENTS

# 03 Why esProc Works Better

# Why SQL is difficult to write: What is the max days has a stock been rising?

```
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 doesn't support ordered operation sufficiently and doesn't provide orderly grouping directly; Instead, four layers of nesting has to be used in a roundabout way.

Such statements are not only difficult to write, but also difficult to understand.

In the face of complex business logic, the complexity of SQL will increase sharply, which is difficult to understand and write.

It isn't an unusual requirement, and it appears everywhere in thousands of lines of SQL in reality, which reduces the efficiency of development and maintenance severely.

# Why can‘t SQL run fast: Get the top 10 from 100 million rows of data

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

This query uses ORDER BY. If it is executed strictly according to this logic, it means sorting the full amount of data, and the performance will be poor.

We know that there is a way to perform this operation without full sorting, but SQL cannot describe it. We can only rely on the optimization engine of the database.

In simple cases (such as this statement), many databases can make the optimization, but if the situation is more complex, the database optimization engine will faint

In the following example, get the TopN from each group, SQL cannot describe it directly, and can only write it as a subquery using window function in a roundabout approach.

In the face of this roundabout approach, the database optimization engine cannot do the optimization and can only perform sorting.

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

# The SPL solution

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

The computing logic of this SPL is the same as that of the previous SQL, but SPL provides orderly grouping operation, which is intuitive and concise.

| | A | |
|---|---|---|
| 1 | =file("Orders.ctx").open().cursor() | |
| 2 | =A1.groups(;top(10;-Amount)) | Top 10 orders |
| 3 | =A1.groups(Area;top(10;-Amount)) | Top 10 orders of each area |

SPL regards TopN as the aggregation operation of returning a set, avoiding full sorting; The syntax is similar in case of whole set or groups, and there is no need to use the roundabout approach.

# Why is SPL more advanced?

【Analogy】Calculate1+2+3+...+100=?

| Ordinary people will do like this | Gauss does like this |
| --- | --- |
| 1+2=3<br>3+3=6<br>6+4=10<br>10+5=15<br>15+6=21<br>21+7=28<br>... | 1+100=101<br>2+99=101<br>...<br><br>**A total of fifty 101**<br>50*101= 5050 |

The smart Gauss came up with a convenient and efficient algorithm.

The key here is:

**Multiplication** used!

*Extended reading:*
*SPL: a database language featuring easy writing and fast running*

SQL is like an arithmetic system with only **addition**. The code is lengthy and the calculation is inefficient.

SPL is equivalent to the invention of **multiplication**! Simplify writing and improve performance.

The difficulties of SQL stem from relational algebra, and theoretical problems cannot be solved by engineering methods. Despite years of improvement, it is still difficult to meet complex requirements.

SPL is based on a completely different theoretical system: discrete dataset. SPL provides more abundant data types and basic operations, and has more powerful expression capabilities.

# Common scenarios to beat SQL

## 1. Complex orderly Computing: funnel analysis of user behavior transformation

- Calculate the user churn rate after each event (page browsing, search, shopping cart addition, order placement, payment, etc.)
- Multiple events are effective only when they are completed within a specified time window and occur in a specified order. It is very hard to implement it in SQL, not to mention optimize it.

## 2. Multi-step big data batch job

- Complex business requirements are difficult to implement directly in SQL, cursor reading is slow and difficult to calculate in parallel, wasting computing resources
- The implementation using stored procedure requires thousands of lines and tens of steps, with the repeated buffering of intermediate results, the batch job cannot be completed within the specified time window

## 3. Multi index calculation on big data, repeated use and multiple associations

- Perform the calculation of hundreds of indexes at one time, and use the detailed data for many times, and association is also involved. SQL needs to traverse data repeatedly.
- Mixed calculation of large table association, conditional filtering, grouping and aggregation, and deduplication; Real-time calculation with high concurrency.

In real business, complex SQL (and stored procedures) are often hundreds/thousands of lines, and a large number of roundabout approaches have to be used to implement the calculation. The code becomes complex as well as the performance becomes low.

# Funnel analysis of an E-commerce company

```
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 provides order-related calculations and is more thoroughly set-oriented. Code is written directly according to natural thinking, which is simple and efficient.

This code can handle funnels with any number of steps, as long as the parameters are changed.

SQL lacks order-related calculations and is not completely set-oriented. It needs to detour into multiple subqueries and repeatedly JOIN. It is difficult to write and understand, and the operation performance is very low.

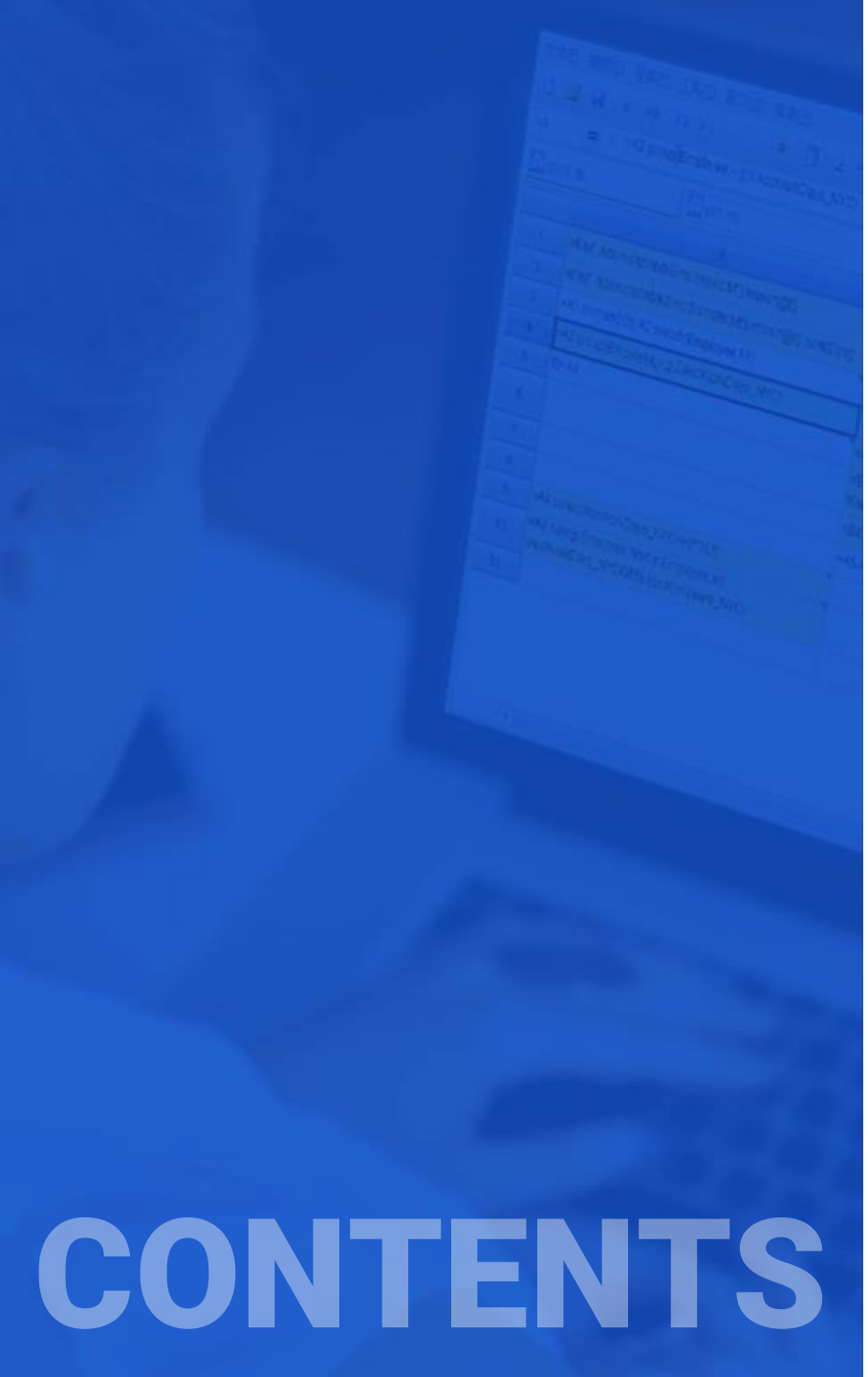
Due to space limitation, only a three-step funnel is listed here, and subqueries need to be added when adding more steps.

# Part of High Performance Computing Mechanism Provided by SPL

| Traversal technique | Highly efficient Joins | High performance storage | Cluster computing |
|---|---|---|---|
| Delayed cursor | ※ Foreign key as pointer | Orderly Compressed Storage | Preemptive Load Balancing |
| Aggregate Understanding | ※ Numbering of foreign keys | Free column storage | ※ Multi-zone composite table |
| ※ Ordered cursor | Order-based merge | ※ Hierarchical Numbering positioning | ※ Cluster dimension table |
| ※ Multi-purpose traversal | ※ Attached table | Index and Caching | ※ Memory spare tire fault tolerance |
| Prefilter traversal | ※ Unilateral HASH Join | ※ Double increment segmentation | External storage redundancy fault tolerance |

※ Many algorithms and storage schemes here are the original inventions of SPL!

*Extended reading: Performance Optimization for Big Structured Data*

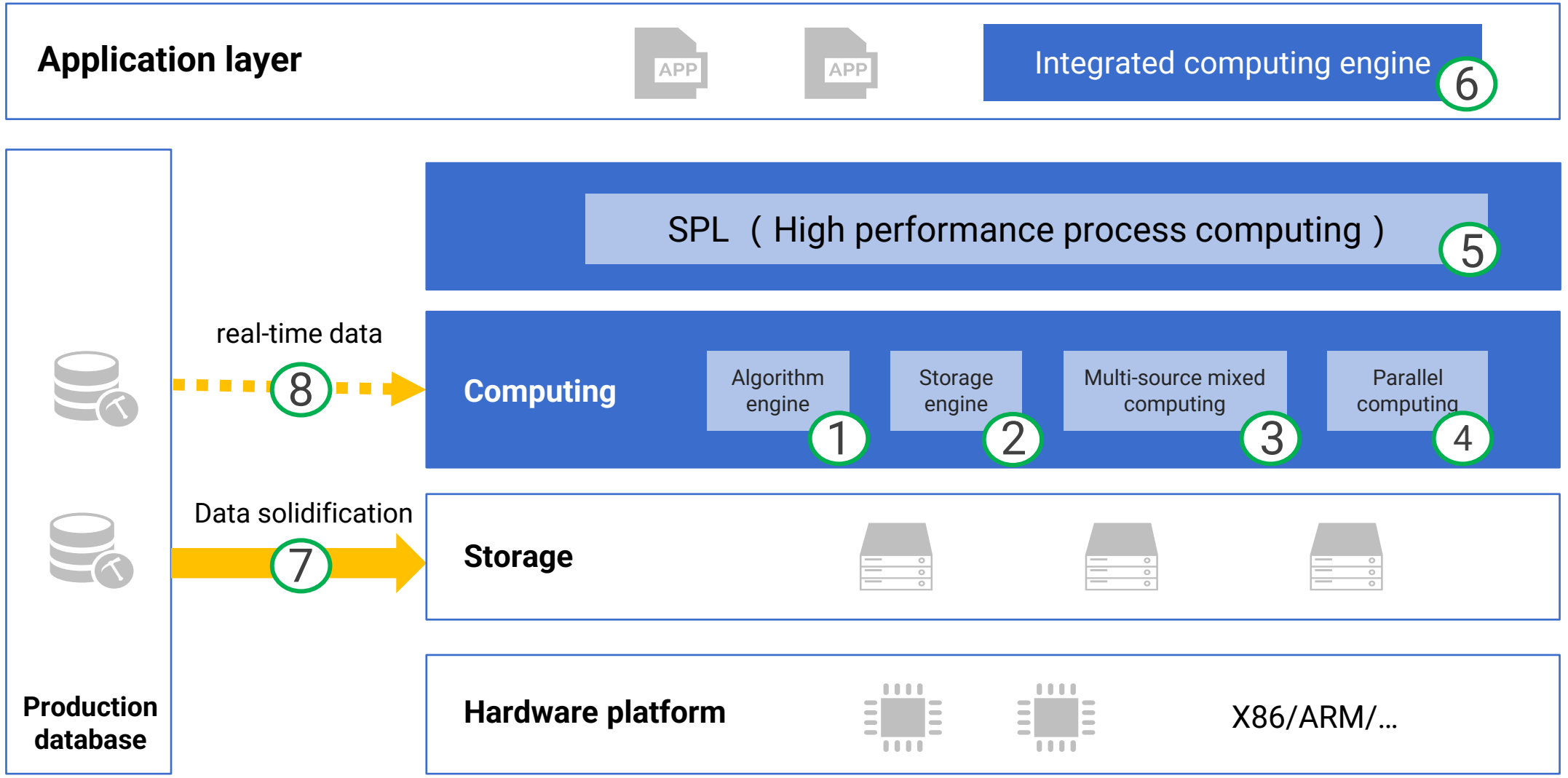


CONTENTS

04

# Technical Characteristics

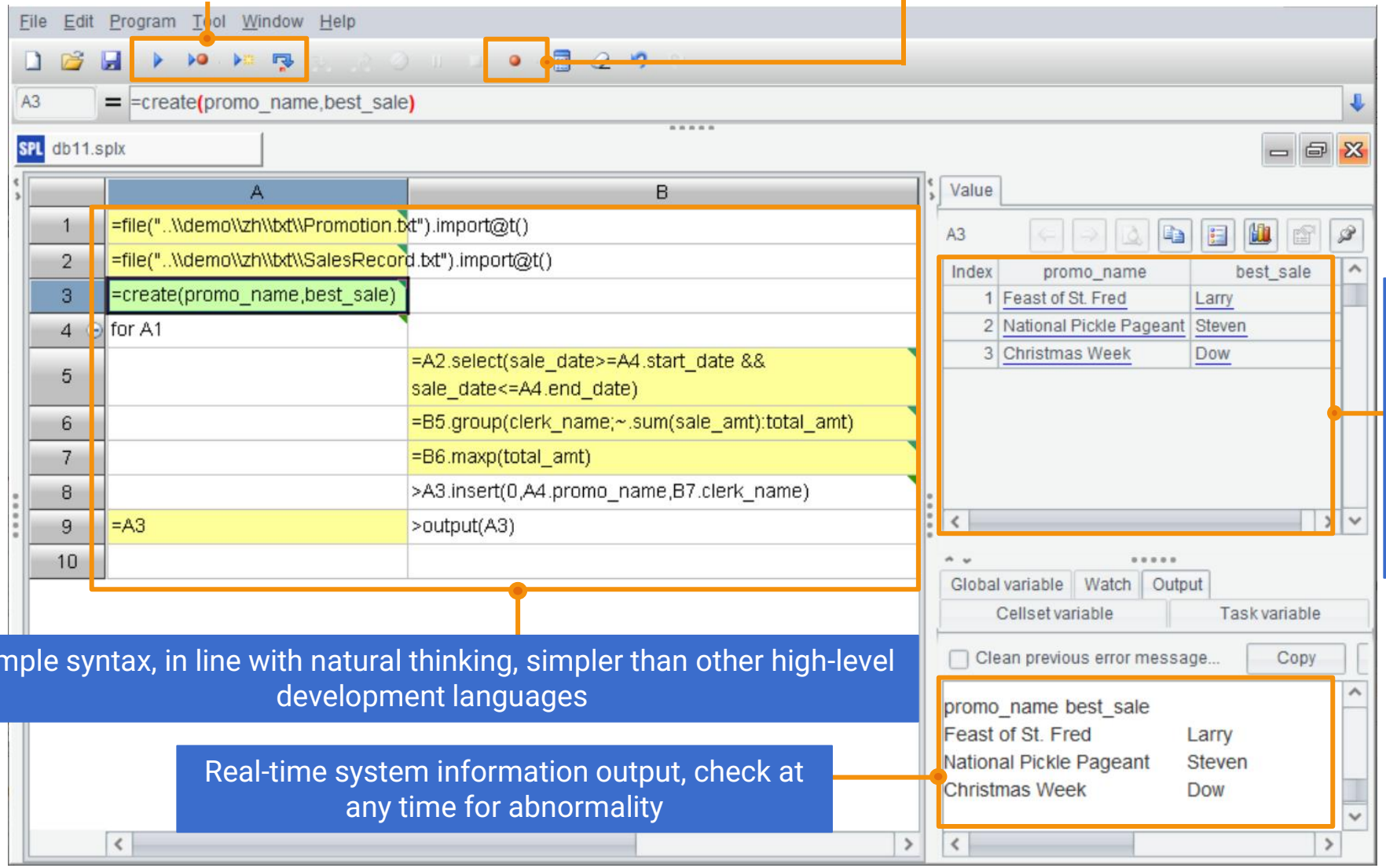
# esProc Technical Architecture



# Simple and easy-to-use development environment

Run, debug, run to cursor, step

Set breakpoint




What you see is what you get cell result ,easy to debug; Convenient to reference intermediate result

Simple syntax, in line with natural thinking, simpler than other high-level development languages

Real-time system information output, check at any time for abnormality

# Specially designed syntax system

SPL is especially suitable for complex process operations.

| | 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) | | | | |

**Natural & clean step-by-step computation**, direct reference of cell name **without specifically defining a variable**

# Rich class libraries

Designed specifically for structured data tables

| | A | B | C |
|---|---|---|---|
| 1 | =esProc.query("SELECT | orderID as contract, | /retrieve sales records |
| 2 | =A1.group(salesman) | | |
| 3 | =create(salesman,thisyear | Amount, 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)) | |

**Grouping & Loop**

| | 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 | | | |

**Sorting & Filtering**

| | 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) | | |

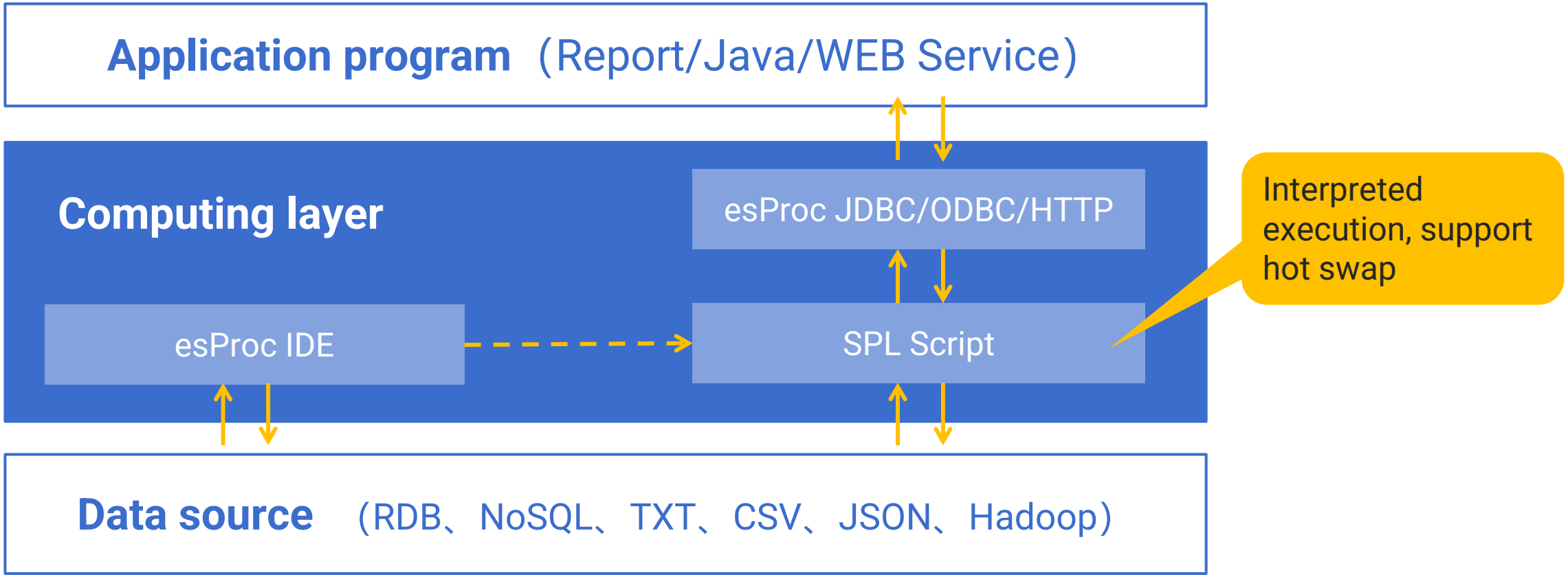
**Set operations**

| | 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) | | |

**Ordered sets**

# Excellent integration

esProc, developed in Java, can run independently or be seamlessly integrated into applications.



# Support diverse data sources

Multiple data sources are directly used for mixed computations.


Direct computation without loading data into database, using the strengths of original data sources, real time computation

# Flexible and efficient file storage

## High performance

Private data storage format: bin files, composite tables

## File system

Support to store data by business classification in tree directory

**BIN FILE**

Double increment segmentation supports arbitrary number parallelism
Self-owned efficient compression technique (reduced space; less CPU usage; secure)
Generic Storage, allowing set data



**COMPOSITE TABLE**

**Mixed row and column** storage

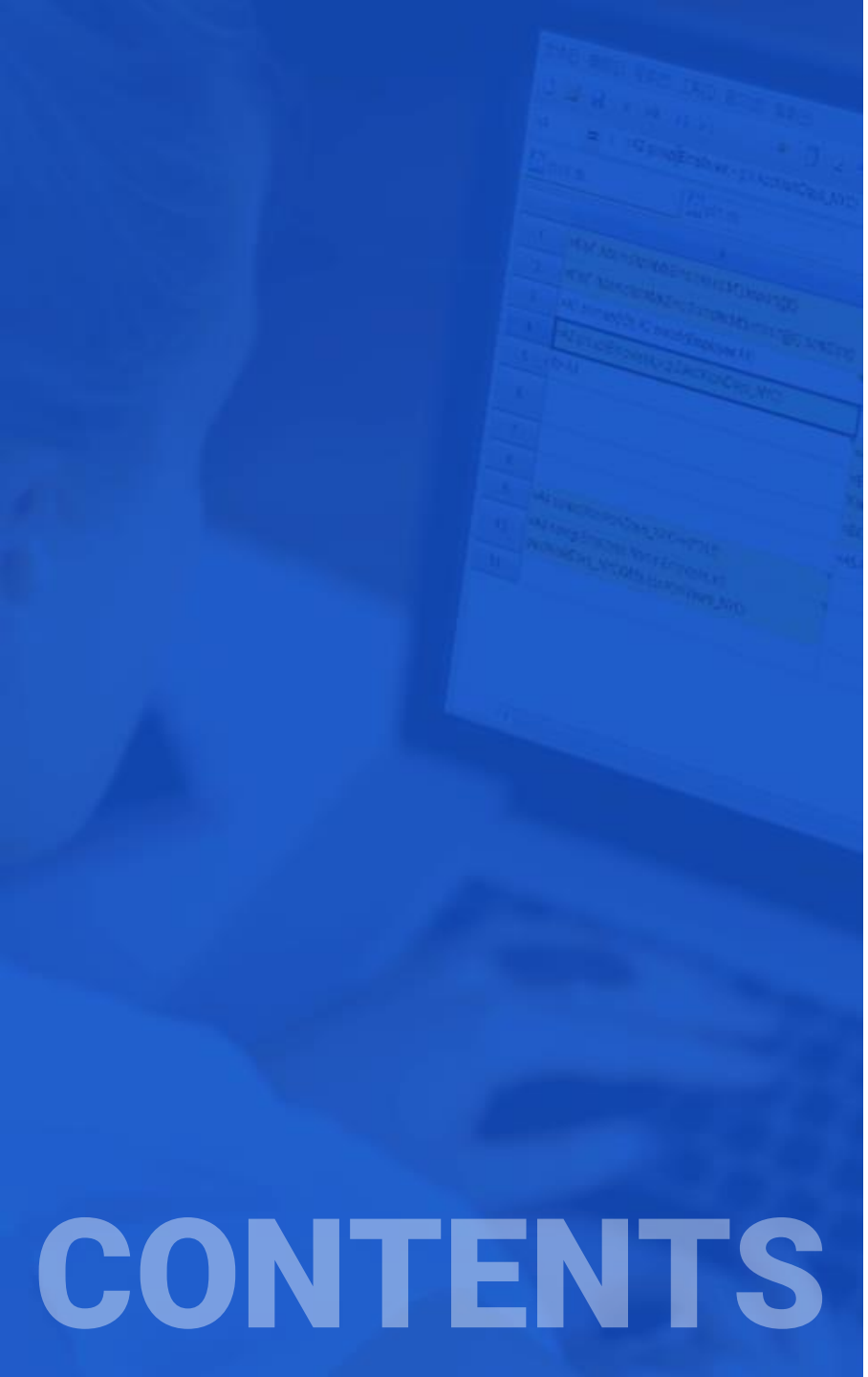
**Ordered storage** improves compression rate and positioning performance

Efficient **intelligent index**

**Double increment segmentation** supports arbitrary number parallelism
**Integration of main and sub table** to reduce storage and association
**Numbering keys** to achieve efficient positioning Join



Direct file storage without database is more efficient, more flexible, and cheaper.



CONTENTS

05

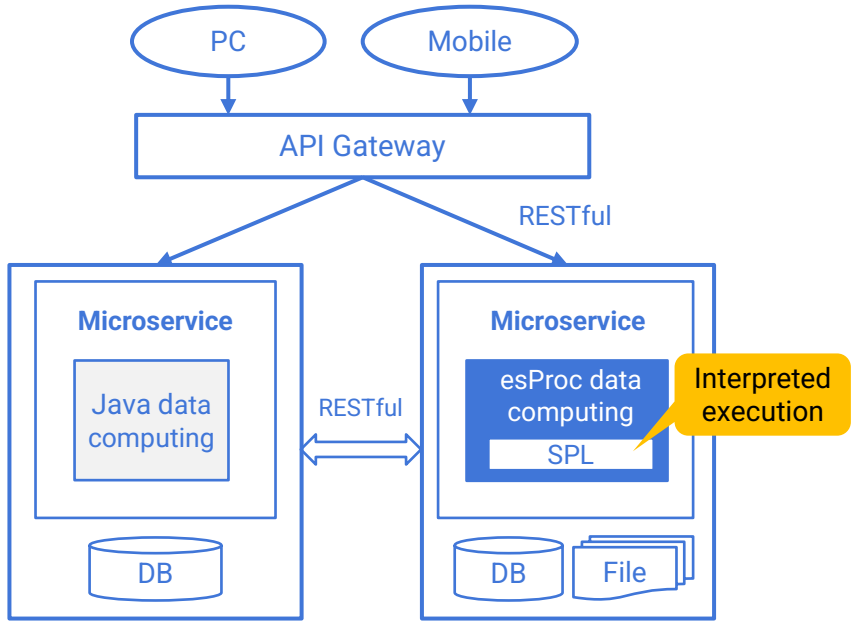
# More Solutions

# Implementation of data-driven micro-service

**Problems**

- Mainstream frameworks such as microservices require data processing at the application side.
- The database is difficult to be embedded in front-end applications, thus hardcoding is the only choice.
- Java/ORM lacks sufficient structured computing class library, which makes it difficult to develop data processing, and hot swap can not be achieved.

**Solution**




- ✓ SPL replaces Java/ORM to implement data computing in (micro-service) applications.
- ✓ Rich class library and agile syntax simplify the development.
- ✓ The system is open and can process data of any source in real time.
- ✓ SPL is interpreted executed, naturally supporting hot swap.
- ✓ Efficient algorithms and parallel mechanism ensure computing performance.

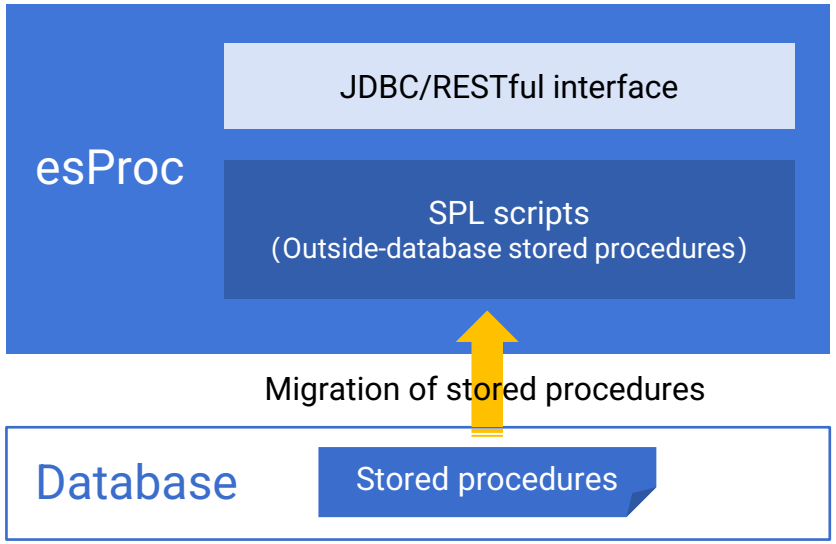
Extended reading : http://c.raqsoft.com/article/1647501354587

# Replace stored procedures

**Problems**

- Stored procedures are hard to edit and debug, and lack migratability.
- Compiling stored procedures requires high privilege, causing poor security.
- The shared use of a stored procedure by multiple applications will cause tight coupling between applications.

**Solution**




- ✓ SPL is intuitively suitable for complex multi-step data computation.
- ✓ SPL scripts are naturally migratable.
- ✓ The script only requires the read privilege of the database and will not cause database security problems.
- ✓ Scripts of different applications are stored in different directories, which will not cause coupling between applications.

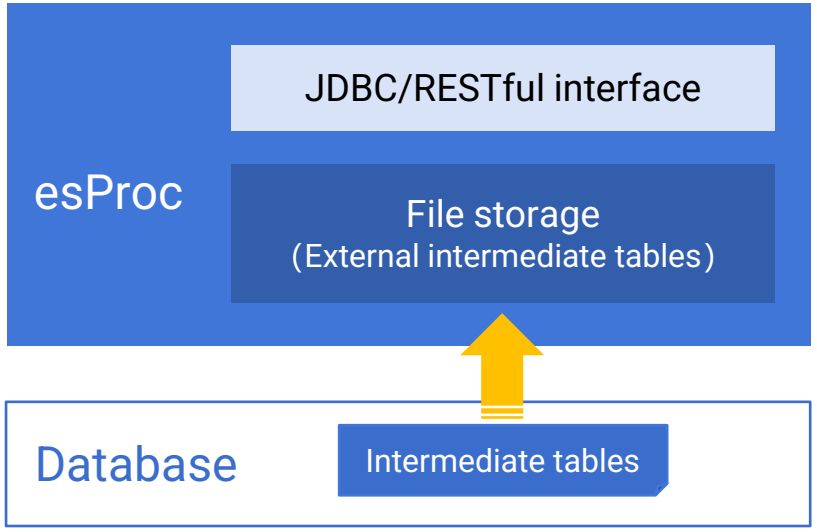
Extended reading: http://c.raqsoft.com/article/1646637335078

# Eliminate intermediate tables from databases

**Problems**

- For query efficiency or simplified development, a large number of intermediate tables are generated in the database.
- The intermediate tables take up large space, causing the database to be excessively redundant and bloated.
- The use of the same intermediate table by different applications will cause tight coupling, and it is difficult to manage the intermediate tables (hard to delete).

**Solution**




- ✓ The aim for storing intermediate tables in the database is to employ the database's computational ability for subsequent computations; SPL can implement the subsequent computations after using file storage.
- ✓ External intermediate tables (files) are easier to manage, and using different directories for storage will not cause coupling problems between applications.
- ✓ External intermediate tables can fully reduce the load on the database, even without the need to deploy the database.

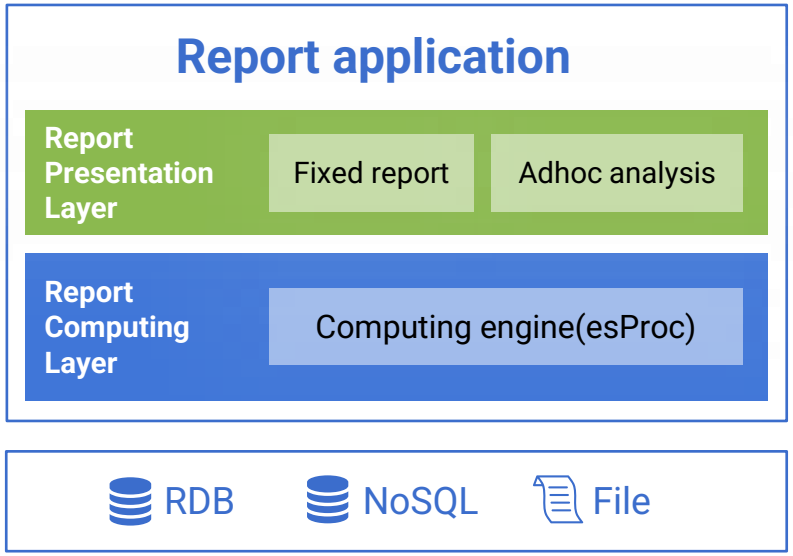
Extended reading: http://c.raqsoft.com/article/1652163420524

# Handle endless report development requirements

**Problems**

- Reporting tools/BI tools can only solve the problems in the report presentation stage and can do nothing about data preparation.
- Data preparation implemented in SQL/stored procedure/Java hardcoding is difficult to develop and maintain, and the cost is high.
- The report development needs are objectively endless, and data preparation is the main factor leading to high development costs.

**Solution**



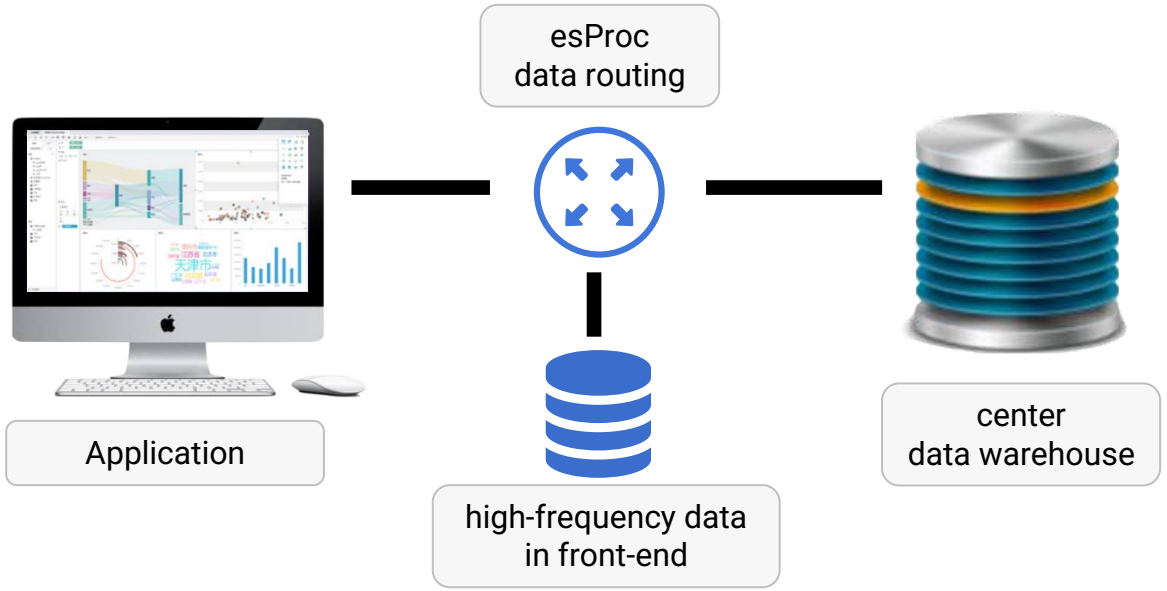

- ✓ Add a computing layer between report presentation and data source to solve the data preparation problems.
- ✓ SPL simplifies the data preparation of reports, makes up for the lack of computing ability of reporting tools, and comprehensively improves the efficiency of report development.
- ✓ Both report presentation and data preparation can quickly respond to handle endless report development needs at low cost.

Extended reading: http://c.raqsoft.com/article/1642764157224

# Programmable data routing to implement front-end calculation

Center data warehouse needs front-end calculation to share the heavy workload

- Move only high frequency data to front, take over most of the calculation requests
- The programmable data routing automatically selects the front-end data and data warehouse and mixes the calculation results, applications have transparent access to full data



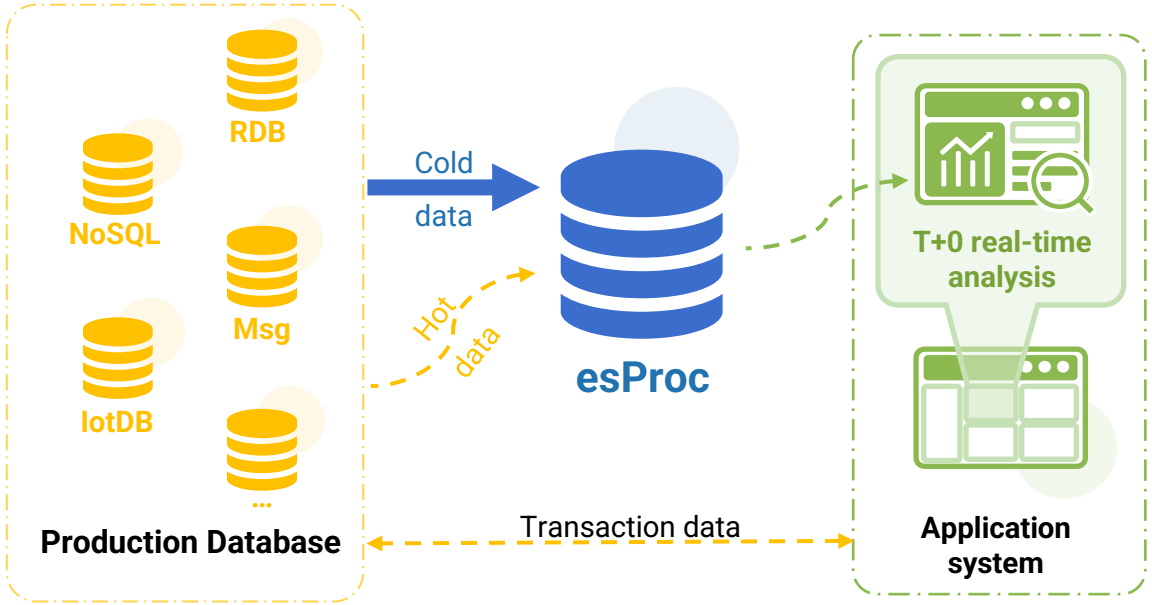

Data routing enables low-cost high-performance full data analysis

- **Dilemmas of implementing front-end calculation through traditional database**
  - Application may access all the data, and moving full data to front causes redundant construction and high cost
  - Due to lack of data routing, moving part of the data to front causes that application can not transparently access the full data and the poor experience

Extended reading: http://c.raqsoft.com/article/1672364811904

# Mixed computation to implement real-time HTAP

- **Mixed calculation of hot and cold data to implement T+0 real-time analysis**
  - Organized historical cold data
  - Real time reading of transaction hot data
- **The production system needs no modification**
  - Make full use of the advantages of the original multiple data sources



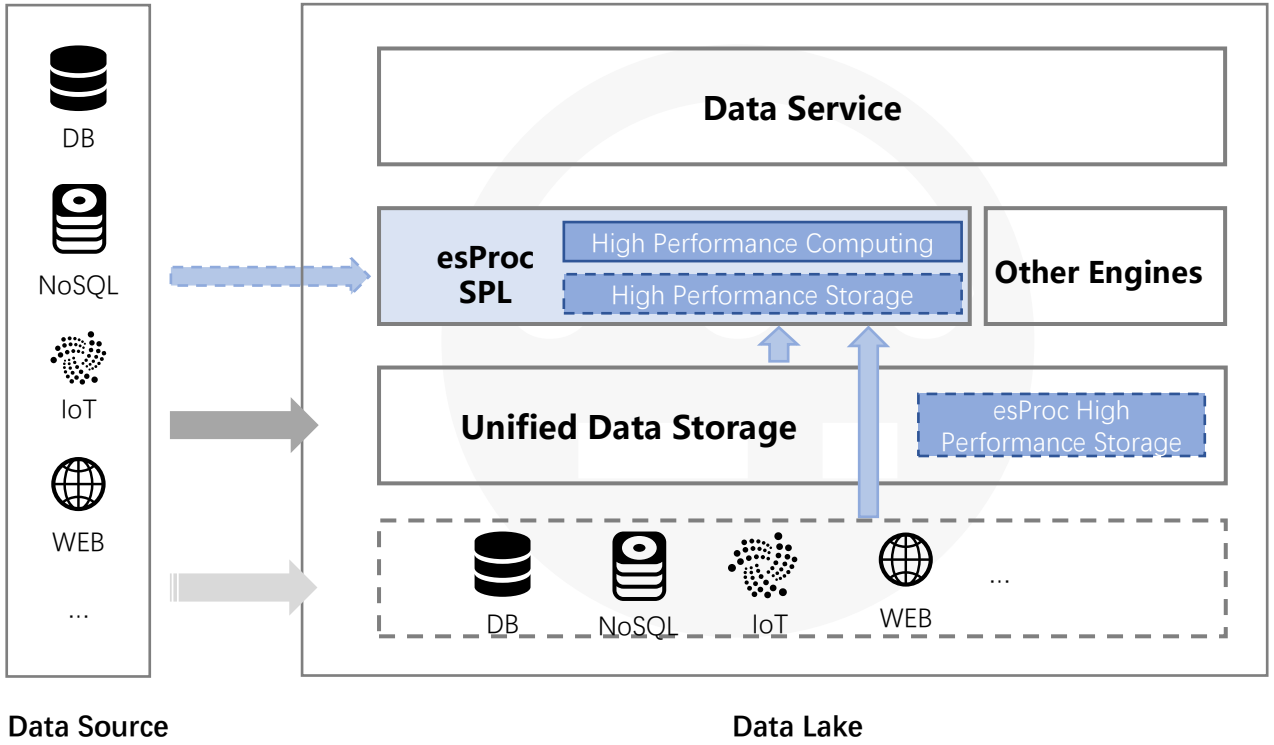

**Support low-risk, high-performance and strong real-time HTAP with open multi-source hybrid computing capability**

- **HTAP databases are difficult to meet HTAP requirements**
  - Requiring replacement of production system database, with high risk
  - Insufficient SQL computing power, insufficient historical data preparation, low performance
  - Closed computing capacity, complex ETL process required due to external diverse data sources, poor real-time ability

Extended reading: http://c.raqsoft.com/article/1666750912264

# Perform computation on files to implement Lakehouse

- Open format file data computation
  - txt/csv/xls/json/xml
- High performance private format file storage and computation
- Enter at will and sort out gradually； Lake to House
- Rich data source interfaces, direct real-time computing




- RDB can only House, not Lake
- Strong constraints, non-compliant data cannot enter, and complex ETL processes are inefficient
- Closed, external data cannot be calculated, let alone mixed real-time calculation

Extended reading： http://c.raqsoft.com/article/1667439960234

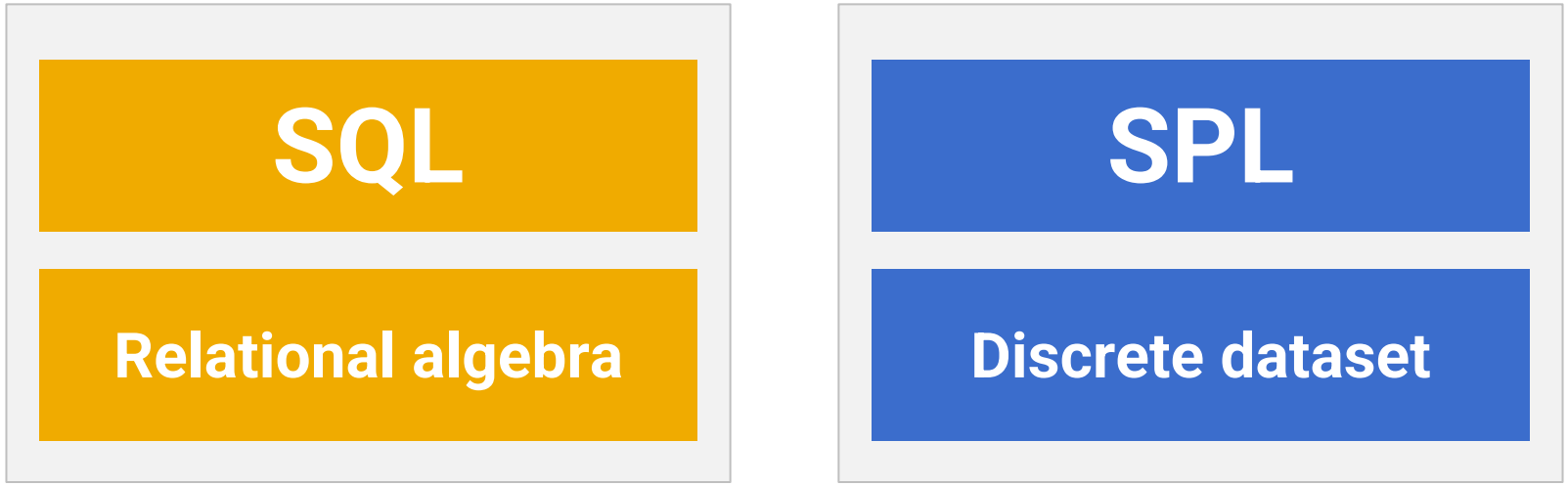
# FAQ

## Frequently asked questions

# Q Is esProc based on open source or database technology?

A

esProc is based on a brand-new computing model, no open source technology can be cited, and all independent innovation from theory to code.




SPL is based on innovation theory that can no longer use SQL to achieve high performance, and SQL can not describe most low complexity algorithms.

## Q Where can esProc be deployed

A

esProc is implemented in pure Java.

esProc can run smoothly under any OS equipped with JVM, including VM, cloud server and even container.

## Q How applications invoke esProc?

A

esProc provides a standard JDBC driver for Java applications.

esProc can be integrated in a Java application seamlessly.

esProc can be invoked by a non-Java application via HTTP/RESTFul

## Q Can esProc run based on the existing database?

A

Yes, Of course!

However, esProc can not guarantee high performance in this situation due to the inefficient I/O of database, and database can hardly provide storage schema which is necessary for low complexity algorithm.

## Q Where does esProc store data?

A

esProc stores data in files of private format to guarantee performance.

Any file system installed on OS is available, including NFS.

esProc can easily implement separation between storage and computation to scale out

# What are the weaknesses of esProc?

**Comparing with RDB:**

The metadata is immature in esProc, most of computation will begin from accessing data source, it will be a little tedious for simple operations.

**Comparing with Hadoop/MPP:**

The cluster function of esProc has not many chances to  be well-trained.

esProc has reduced many clusters into a single machine without sacrificing performance in history.

**Comparing with Python:**

SPL is developing its AI functions, but now is still not even close to Python.

## Q How difficult is it to learn SPL

A

SPL is dedicated to low code and high performance.

Learning SPL is not difficult. It can be mastered in hours and skilled in weeks.

What is  difficult is to design optimization algorithms!

## Q Is there a tool to convert SQL to SPL automatically?

A

Not yet.

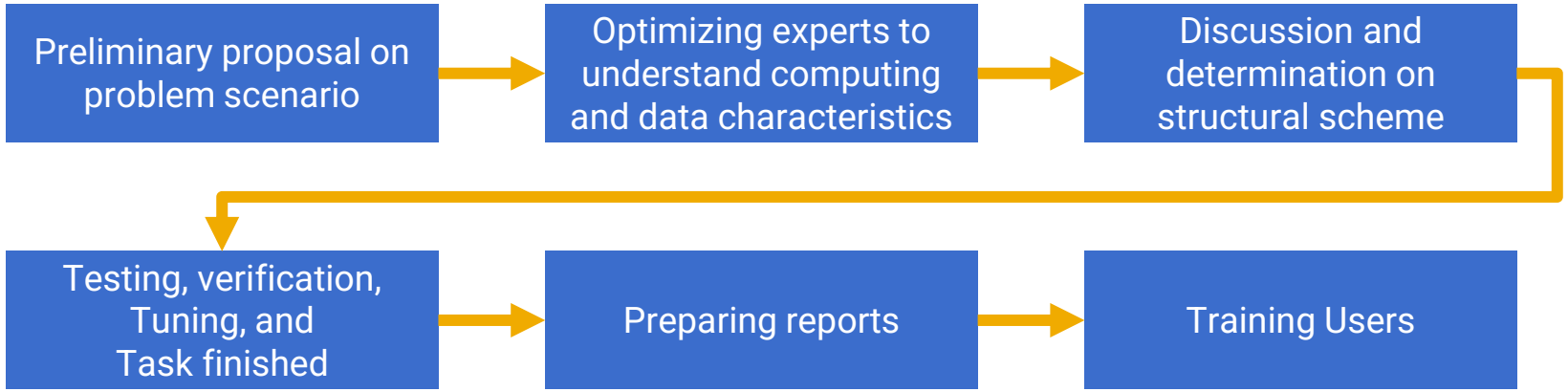
The information in SQL statement is insufficient to optimize its performance. Frankly, we are not a veteran like RDB vendor for guessing goal of a SQL, so converting SQL to SPL directly will usually lead to slower speed.

# How to launch a performance optimization process

**The first 2-3 scenarios will be implemented by Scudata engineer in collaboration with users.**

Most programmers are used to the way of thinking in SQL and are not familiar with high performance algorithms. They need to be trained to understand in one or two scenarios.

Dozens of performance optimization routines will be experienced and learned. Algorithmic design and implementation are not so difficult.




**Give a man a fish and you feed him for a day. Teach him how to fish and you feed him for a lifetime!**

Extended reading： What should we do when SQL (and stored procedure) runs too slowly?

# Summary of advantages of esProc

## High performance

The processing speed of big data is 1 order of magnitude higher than that of traditional solutions

## Efficient development

Procedural syntax, in line with natural thinking

Rich class libraries

**5**

**advantages**

## Flexible and open

Multi-source mixed computation

Can run independently, or embedded into applications

## Save resources

Single machine can match cluster, reducing hardware expense

Environment-friendly

## Sharp cost reduction

Development, hardware, O&M costs reduced by X times


THANKS