SPL

Structured Process Language

SPL

Structured Process Language

A procedure-oriented structured data computing language

CONTENTS

Set orientation

Discreteness

Deep
set orientation

Orderliness

SPL features

Set orientation

Set-oriented

Structured data is always found in batches

set operations · lambda syntax · dynamic data structure

SQL is set-oriented

WHERE,ORDER BY,GROUP

INTERSECT,UNION,MINUS

Discreteness

Members of a set:

Can exist independently;

Can be computed separately, or join up with other separate members to perform a set operation

SQL's non-discreteness:

Permits only single-record tables, but not separate records;

Calculates each operation consistently and can't retain any individual records

Case study

Calculate differences in age and salary in SPL

	A	
1	=employee.select@1(name==“Tom")	
2	=employee.select@1(name==“Harry")	
3	=A1.age-A2.age	
4	=A1.salary-A2.salary	

SQL solution

Calculate differences in age and salary in SQL

```
SELECT (SELECT age FROM employee WHERE name=‘Tom′ )
    - (SELECT age FROM employee WHERE name=‘Harry’) FROM dual
SELECT (SELECT salary FROM employee WHERE name=‘Tom′ )
    - (SELECT salary FROM employee WHERE name=‘Harry’) FROM dual
```

Case study

Update by a certain condition: Resellers ranked among the first 10% will be rewarded 5% of its performance

	A	
1	=agent.sort@z(amount).to(agent.len()*0.1)	Get the first 10% of the resellers in performance
2	=A1.run(amount=amount*1.05)	Offer a reward

The referencing foreign key: Find the records of transaction done in Beijing

	A	
1	>TRANSACTION.switch(AREA,AREA:ID)	Create a referencing foreign key
2	=TRANSACTION.select(AREA.NAME==“Beijing”)	Filter records by the associated table’s key referenced by the foreign key

Deep set orientation

Discreteness is indispensable to deep set orientation;

Discrete records can join up together to form a set

grouping subset · unconventional aggregation · main & sub tables

Grouping subset

Find subject records of students with score totaling over 500

	A
1	=SCORE.group(STUDENTS).select(~.sum(SCORE)>=500).conj()

Discreteness enables pure grouping operation;

Due to the lack of explicit set data type formed by discrete records, SQL can’t help summarizing each grouping subset and thus needs two traversals and joins

1	WITH T AS
2	(SELECT STUDENT FROM SCORE GROUP BY STUDENT HAVING SUM(SCORE)>500)
3	SELECT TT.* FROM T LEFT JOIN SCORE TT on T.STUDENT=TT.STUDENT

Grouping subset

Count User logins in 3 days before last login

	A
1	=LOGIN. group(uid;~.max(logtime):last,~.count(interval(logtime,last)<=3):num)

In SQL it’s hard to write a complex aggregate query over grouping subsets with simple aggregate expression; in SPL it becomes easy with step-by-step computation since subsets can be kept;
SQL needs subqueries, which mean multiple calculations, attached to the original data set

```
WITH T AS
   (SELECT uid,max(logtime) last FROM LOGIN GROUP BY uid)
SELECT T. uid,T.last,count(TT.logtime)
    FROM T LEFT JOIN LOGIN TT ON T.uid=TT.uid
    WHERE T.last-TT.logtime<=3 GROUP BY T.uid,T.last
```

Unconventional aggregation

Find the first login records of users

	A
1	=LOGIN.group(uid).(~.minp(logtime))

Getting a member from a set is also a kind of aggregate operation;

With discreteness, SPL performs such an aggregation directly over the grouping subsets

1	SELECT * FROM
2	(SELECT RANK() OVER(PARTITION BY uid ORDER BY logtime) rk, T.* FROM LOGIN T) TT
3	WHERE TT.rk=1

Unconventional aggregation

Get the interval between last two logins for each user

	A	
1	=LOGIN.groups(uid;top(2,-logtime))	Last two login records
2	=A1.new(uid,#2(1).logtime-#2(2).logtime:interval)	Calculate intervals

An aggregate function can return a result set;

Deep set orientation makes it easy to perform an aggregation returning a set over grouping subsets

1	WITH T AS
2	(SELECT RANK() OVER(PARTITION BY uid ORDER BY logtime DESC) rk, T.* FROM LOGIN T)
3	SELECT uid,(SELECT TT.logtime FROM TT where TT.uid=TTT.uid and TT.rk=1)
4	-(SELET TT.logtim FROM TT WHERE TT.uid=TTT.uid and TT.rk=2) INTERVAL
5	FROM LOGIN TTT GROUP BY uid

Main& sub tables

Calculate amount from order details

	A	
1	=ORDER.derive(OrderDetail.select(ID==ORDER.ID):DETAIL)	Create a record type field for the sub table
2	=A1.new(ID,CUSTOMER,DETAIL.sum(UnitPrice*QUANTITY):AMOUNT)	Calculate order amount

A record type field is suitable to describe multilevel data, including main & sub tables;

Without explicit record data type, SQL is non-discrete and can't reference individual records; a JOIN needs to precede a GROUP operation

1	SELECT ORDER.ID,ORDER.CUSTOMER,SUM(OrderDetail.PRICE)
2	FROM ORDER
3	LEFT JOIN OrderDetail ON ORDER.ID=OrderDetail.ID
4	GROUP BY ORDER.ID,ORDER.CUSTOMER

Orderliness

Order-based computations require deep set orientation and discreteness ; and

They are determined by both data itself and its position

cross-row reference · order-based grouping · position-based access

Order-based computations

Relational algebra inherits the mathematical concept of unordered sets;

Early SQL would generate sequence numbers and perform a JOIN to perform a limited number of order-based computations

Calculate growth rate of stock

```
WITH T AS
        (SELECT rownum,TransactionDate,ClosingPrice
        FROM ( SELECT * FROM STOCK ORDER BY TransactionDate) )
SELECT T1.TransactionDate,T1.ClosingPrice-T2.ClosingPrice
FROM T T1 JOIN T T2 ON T1.rownum=T2.rownum+1
```

SQL2003 standard offers window functions to generate sequence numbers and reference an adjacent row more conveniently

```
SELECT  TransactionDate,ClosingPrice-LAG(ClosingPrice) OVER (ORDER BY TransactionDate) FROM STOCK
```

Cross-row reference

Calculate order amount from details

	A
1	=SALES.sort(PRODUCT,MONTH)
2	=A1.select(if(PRODUCT==PRODUCT[-1],QUANTITY/QUANTITY[-1])>1.1 && AMOUNT/AMOUNT[-1])>1.1))

Ordered sets support cross-row reference;

A SQL window function needs a subquery to realize a cross-row reference; multiple references need multiple window functions

1	WITH T AS
2	(SELECT QUANTITY/LAG(QUANTITY) OVER(PARTITION BY PRODUCT ORDER BY MONTH) r1
3	(SELECT AMOUNT/LAG(AMOUNT) OVER(PARTITION BY PRODUCT ORDER BY MONTH) r2, A.*, FROM SALES A)
4	SELECT * FROM T WHERE r1>1.1 AND r2>1.1

Cross-row reference

Calculate MA of sales in the previous and next months

	A
1	=SALES.sort(MONTH).derive(AMOUNT{-1,1}.avg()):MA)

Cross-row references apply more easily to ordered sets;

SQL window functions support only the simplest cross-row references, and, if a set is referenced, need to piece together one

```
WITH B AS
        (SELECT LAG(AMOUNT) OVER (ORDER BY MONTH) f1, LEAD(AMOUNT) OVER (ORDER BY MONTH) f2,   A.* FROM SALES A)
        SELECT MONTH,AMOUNT,
        (NVL(f1,0)+NVL(f2,0)+AMOUNT)/(DECODE(f1,NULLl,0,1)+DECODE(f2,NULL,0,1)+1) MA
FROM B
```

Order-based grouping

Count the baby groups that have at least 5 consecutively born boys/girls

	A
1	=BABIES.sort(BirthDate).group@o(GENDER).count(~.len()>=5)

Besides equi-grouping, the grouping could be order-based;

The order-based grouping is defined on an ordered set; create a new group whenever the grouping field value is changed

```
SELECT COUNT(*) FROM
        (SELECT NumOfChanges FROM
                (SELECT SUM(ChangeValue) OVER ( ORDER BY BirthDate) NumOfChanges FROM
                (SELECT CASE WHEN GENDER=LAG(GENDER) OVER (ORDER BY BirthDate) THEN 0 ELSE 1 END ChangeValue  FROM BABIES) )
        GROUP BY NumOfChanges HAVING COUNT(*)>=5)
```

Order-based grouping

Count the longest consecutive rising days for a stock

	A
1	=STOCK.sort(TransactionDate).group@i(ClosingPrice<ClosingPrice[-1]).max(~.len())

Conditional-controlled order-based grouping

```
SELECT max(ConsecutiveDays)-1 FROM
        (SELECT count(*) ConsecutiveDays FROM
                (SELECT SUM(ChangeValue) OVER ( ORDER BY TransactionDate) NonriseDays FROM
                ( SELECT TransactionDate,
                        CASE WHEN  ClosingPrice>LAG(ClosingPrice) OVER( ORDER BY TransactionDate THEN 0 ELSE 1 END ChangeValue
                FROM STOCK ))
        GROUP BY NonriseDays)
```

Hybrid computation

Find stocks that rise for 3 consecutive days

	A
1	=STOCK.sort(TransactionDate).group(Code)
2	=A1.select((a=0,~.pselect(a=if(ClosingPrice>ClosingPrice[-1],a+1,0):3))>0).(Code)

A computation involving both grouping subsets and the order

```
WITH A AS
    (SELECT Code,TransactionDate, ClosingPrice-LAG(ClosingPrice) OVER (PARITITION BY Code ORDER BY GrowthRate) FROM STOCK)
B AS
    (SELECT Code,
        CASE WHEN GrowthRate>0 AND
            LAG(GrowthRate) OVER (PARTITION BY Code ORDER BY TransactionDate) >0  AND
            LAG(GrowthRate,2) OVER PARTITION BY Code ORDER BY TransactionDate) >0
            THEN 1 ELSE 0 END 3-DayConsRiseValue  FROM A)
SELECT distinct Code FROM B WHERE 3-DayConsRiseValue=1
```

Position-based computation

Calculate median price of products

	A
1	=PRICES.sort([Price).([(PRICES.len()+1)\2,PRICES.len()\2+1]).avg()

Access a member of an ordered set by its sequence number;

SQL needs to generate sequence numbers for un unordered set, and the non-stepwise style makes computation even more difficult

```
WITH N AS (SELECT COUNT(1) FROM PRICES)
SELECT AVERGE(PRICE) FROM
        (SELECT PRICE,ROW_NUMBER() OVER (ORDER BY PRICE) r FROM PRICES ) T
        WHERE r=TRUNC((N+1)/2) OR r=TRUNC(N/2)+1)
```

Position-based access

Find a stock’s average growth rate in the 3 days with the highest prices

	A
1	=STORCK.sort(TransactionDate)
2	=A1.calc(A1.ptop(3,-ClosingPrice),ClosingPrice-ClosingPrice[-1]).avg()

Ordered sets support various position-based accesses;

Unordered sets don’t support position-based access, resulting in complex query with more computations

```
SELECT AVG(GrowthRate) FROM
        ( SELECT TransactionDate, ClosingPrice-LAG(ClosingPrice) OVER ( ORDER BY TransactionDate) GrowthRate FROM StockPrice
                WHERE TransactionDate IN
                        (SELECT TransactionDate FROM
                                (SELECT TransactionDate, ROW_NUMBER() OVER(ORDER BY ClosingPrice DESC) Rank FROM STOCK)
        WHERE Rank<=3 )
```

Summary

Discreteness + set orientation

Set orientation is essential to batch processing;

Discreteness generates deep set orientation and enables order-based set computations

Discrete set model = Set operations + Discrete members

=> Deep set orientation/Ordered sets

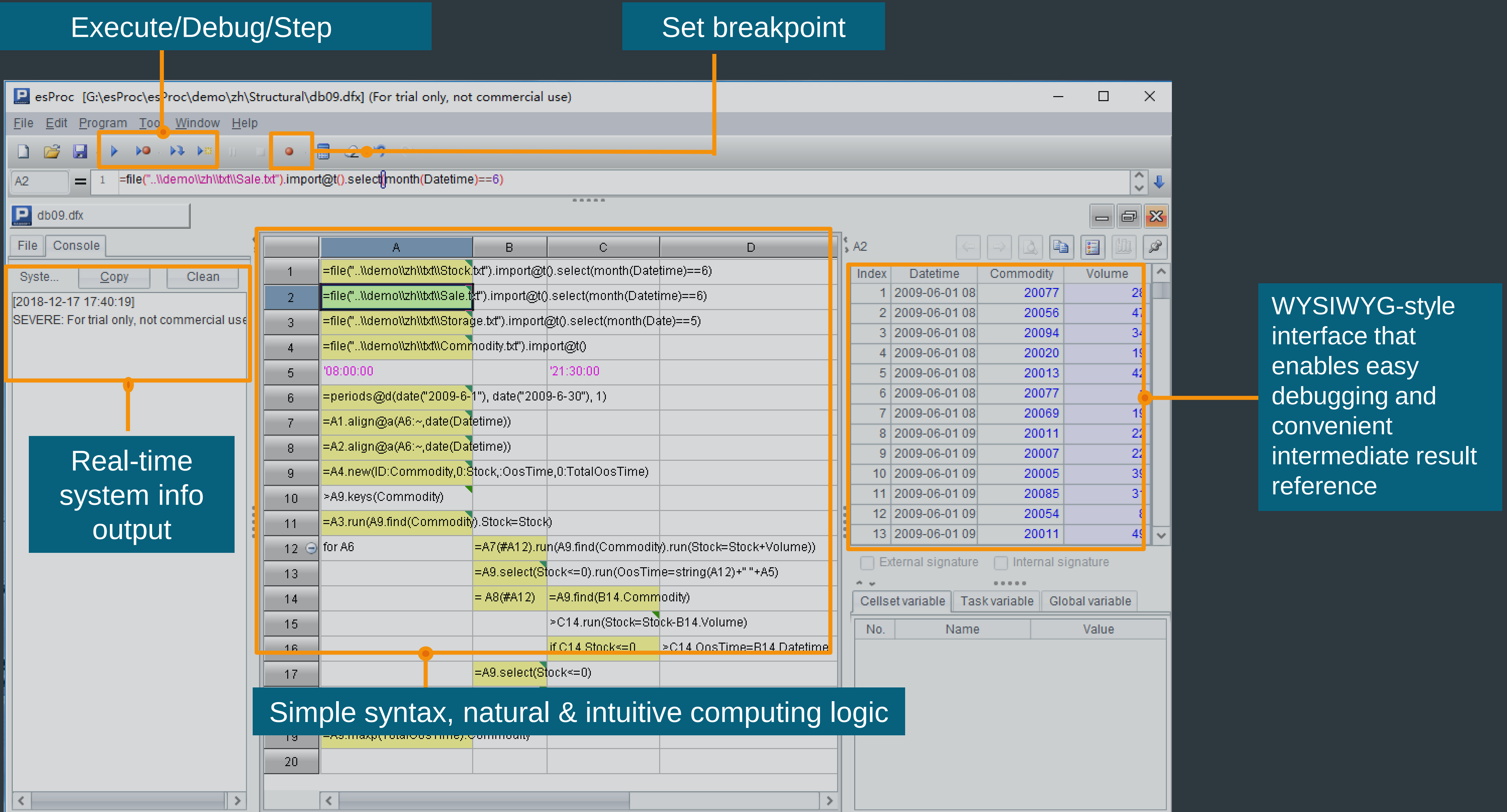
CONTENTS

esProc

esProc is a discrete-data-set-based software product;

SPL is its format language

Development environment



Specially-designed syntax

Particularly suitable for performing complex computations

	A	B	C	D
1	=demo.query("SELECT ORDERID AS CONTRACT,CLIENT,SELLERID AS SALE,AMOUNT,ORDERDATE AS DATE FROM SALES")			
2	=demo.query("SELECT * FROM EMPLOYEE")	=Year=2012		
3	>A1.run(SALE=A2.select@1 (EID:A1.SALE))	/Field Value is Record		
4	=A1.group(SALE)			
5	=create(Sale,ThisYear,LastYear,GrowthRate,NumOfClients,NumOfBigClients,RatioOfBigClients)			
6	for A4	=A6(1).SALE.NAME		
7		=A6.select(year(DATE)==Year).sum(AMOUNT)	/Sales Amount This year	
8		=A6.select(year(DATE)==Year-1).sum(AMOUNT)	/Sales Amount Last Year	
9		=B8/B7-1		
10		=A6.group(CLIENT).(~.sum(AMOUNT))		
11		=B10.count()	/NumbersOf Clients	
12		=B10.count(~>=10000)	/NumbersOf BigClients	
13		=B12/B11		
14		>A5.insert(0,B6,B7,B8,B9,B11,B12,B13)		
15	result A5			

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

Rich class library

Intended for computing structured data

Grouping & Loop

	A	B	C
1	=esProc.query("SELECT OrderID AS Contract,OrderDate AS		/Retrieve Orders table
2	=A1.group(Seller)		
3	=create(Seller,Sales(This year),Sales (Last year),CustomerNumber,BigCustomerNumber)		
4	for A2	=A4(1).Seller	
5		=A4.select(year(Date)==Year).sum(Amount)	
6		=A4.select(year(Date)==Year-1).sum(Amount)	
7		=A4.group(Customer).(~.sum(Amount))	
8		=B7.count()	
9		=B7.count(~>=10000)	
10		>A3.insert(0,B4,B5,B6,B8,B9)	
11	result A3		

Sorting & Filtering

	A	B	C
1	=file("Transaction.txt").import@t()		
2	=A1.sort(CustomerID,TransactionDate)		
3	=A2.select(CarType=="Jetta" || CarType=="Magotan").dup@t()		
4	=A3.derive(interval(TransactionDate[-1],TransactionDate):Interval)		
5	=A4.select(CarType=="Jetta" && CarType=="Magotan" && CustomerID==CustomerID[-1])		
6	=A5.avg(Interval)		
7			
8			
9			
10			
11			

Set operations

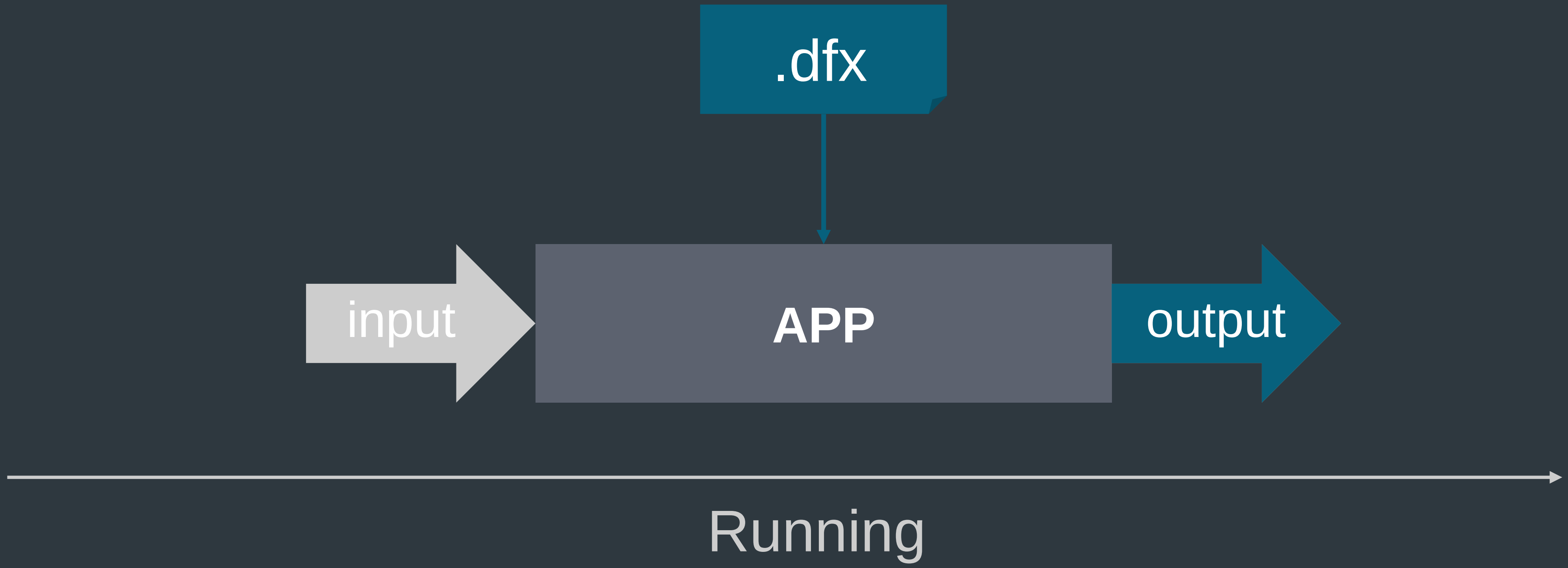
	A	B	C
1	=esProc.query("select * from Employees")		
2	=A1.select(Gender=="Male")		
3	=A1.select(BirthDate>=date("1970-01-01"))		
4	=A2^A3	/Intersection;find the male employees who were born after 1970	
5	=A2&A3	/Union; find the male employees or the employees who were born after 1970	
6	=A2\A3	/Difference; find the employees who were born before 1970	
7	=A4.sum(Wages)		
8	=A5.avg(Age)		
9	=A6.sort(BirthDate		
10	/Set is a widely-used, basic data type		
11			

Ordered sets

	A	B	C
1	=esProc.query("select * from Employees")		
2	=A1.sort(EntryDate)		
3	=A2.pmin(BirthDate)	/Sequence number of the record of the oldest employee	
4	=A2(to(A3-1))	/Access a record with the sequence number	
5	=esProc.query("select * from StockPrice table where StockCode='000062'")		
6	=A5.sort(TransactionDate)		
7	=A6.pmax(ClosingPrice)	/Sequence number of the record with the highest closing price	
8	=A6.calc(A7,ClosingPrice/ClosingPrice[-1]-1)		
9			
10	/Access a record with the sequence number		
11			

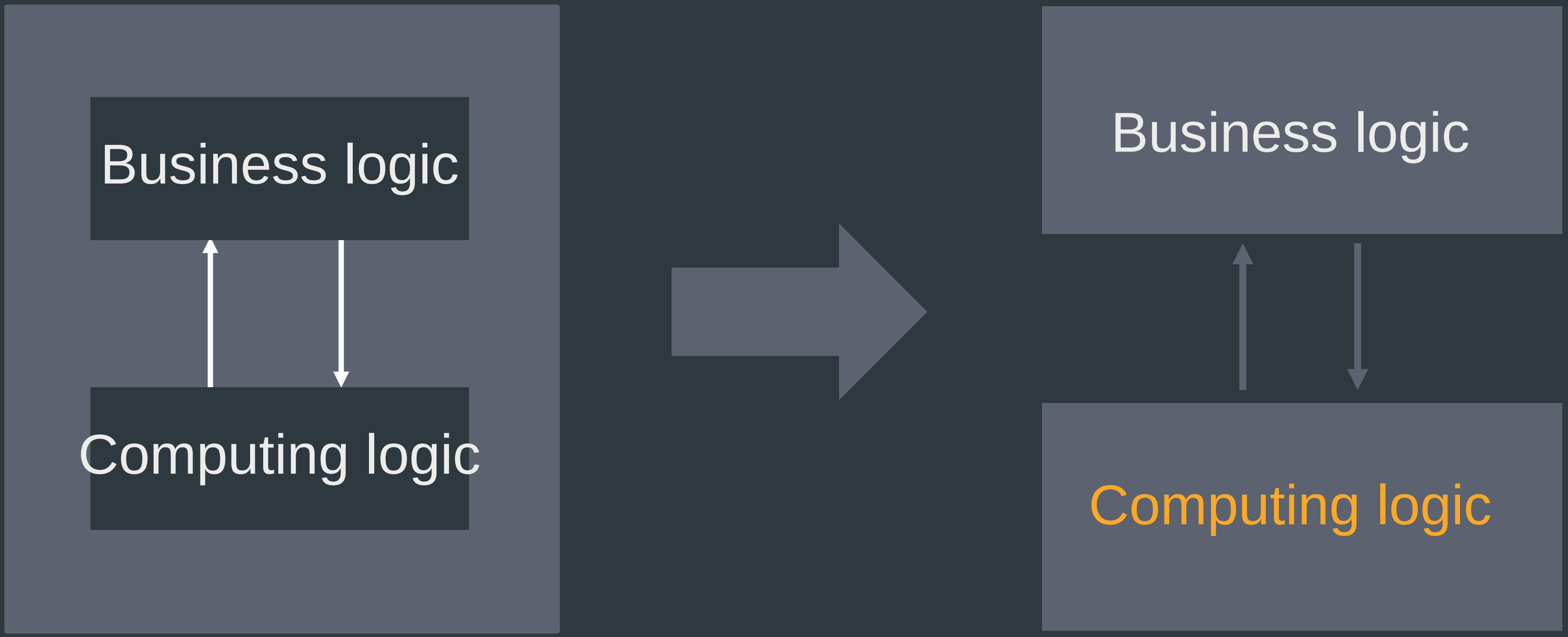
Hot switch

Nonstop switching mechanism in interpreting and executing a script



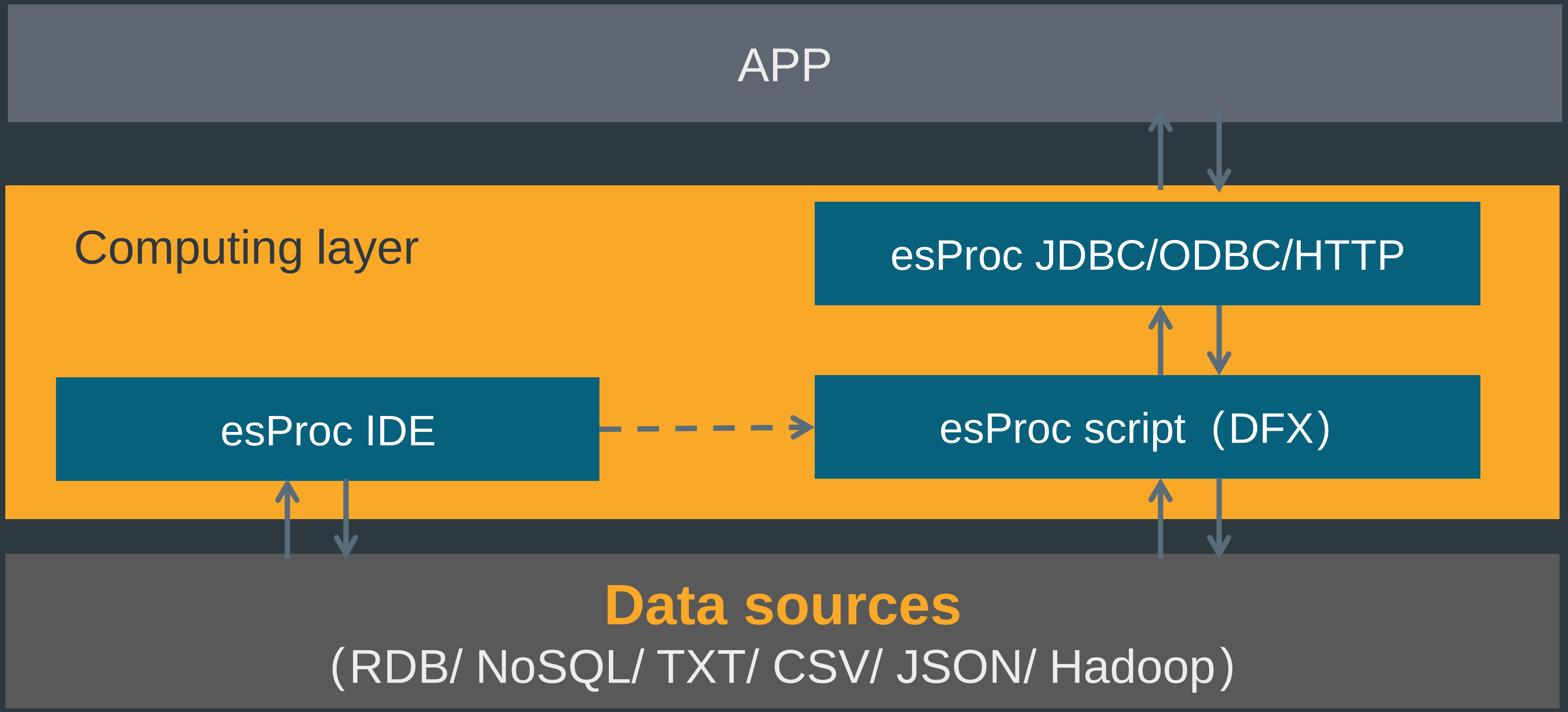
Loose coupling

Script: separate storage & maintaining to achieve modularization



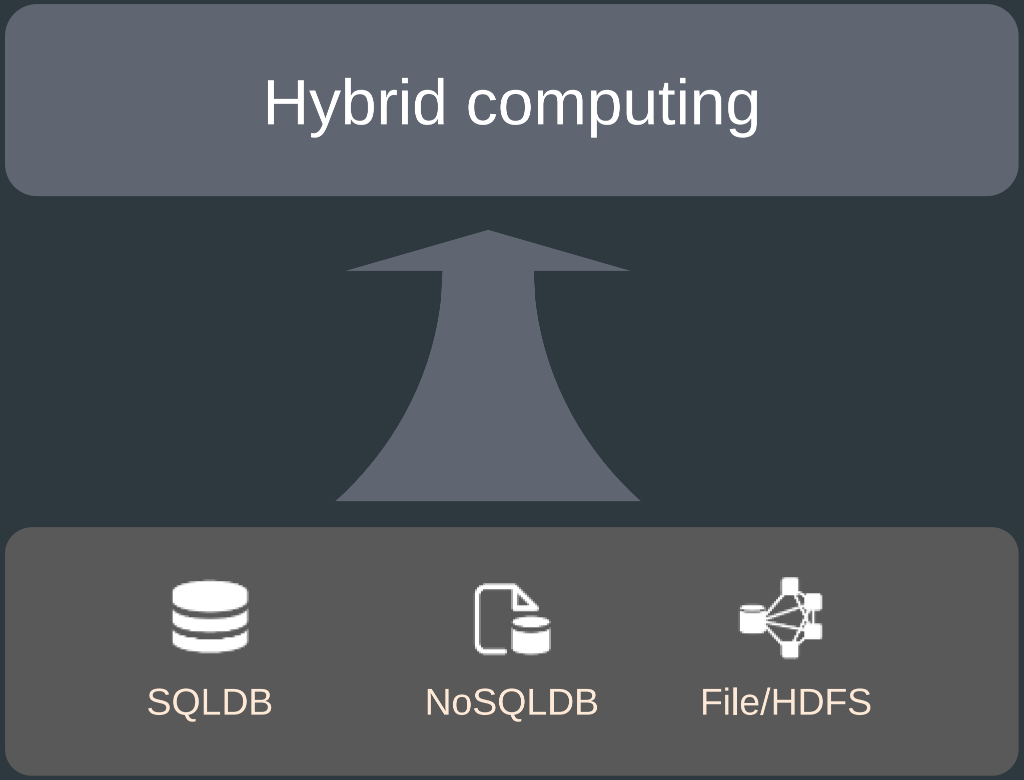
Integration-friendly

Developed in Java, esProc provides standard interface to be seamlessly integrated with a third-party application



Heterogeneous data sources

esProc directly computes data from heterogeneous sources without the need to performing ETL



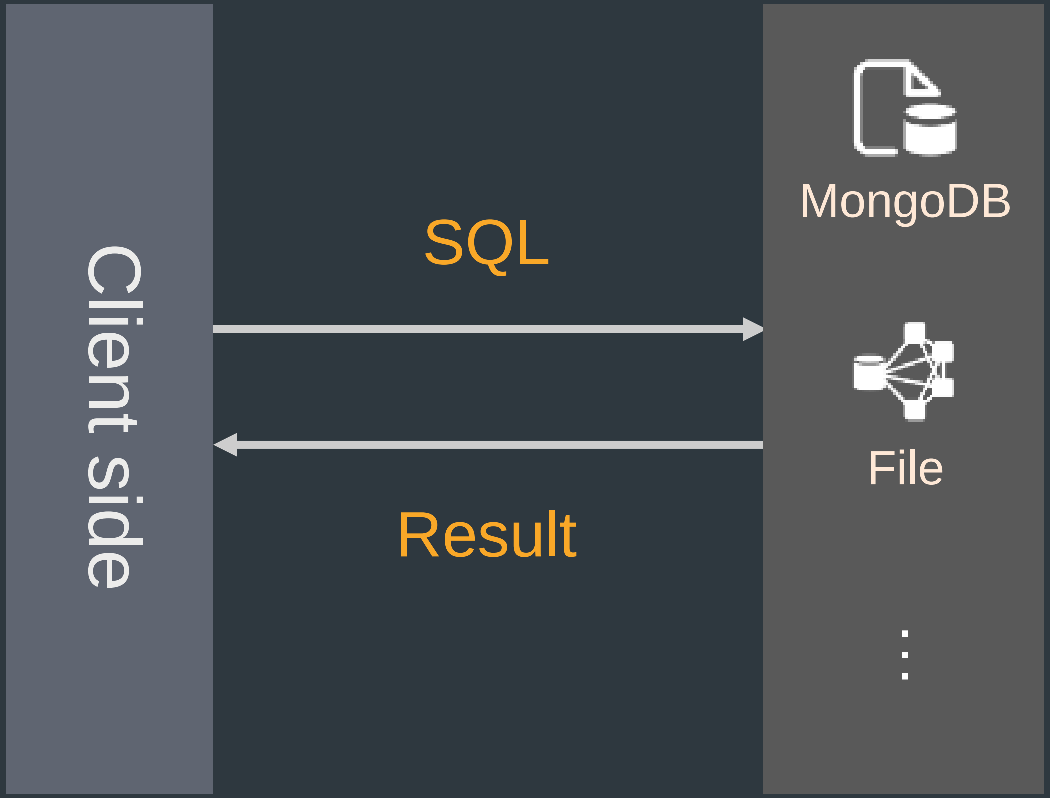
Data interface

- RDB： Oracle,DB2,MS SQL,MySQL,PG,….
- TXT/CSV， JSON/XML， EXCEL
- Hadoop： HDFS， HIVE， HBASE
- MongoDB， REDIS， …
- HTTP、ALI-OTS
- …

Built-in and ready-to-use

SQL query over files

Query NonSQL & files in SQL



Enable SQL queries over NoSQL & files

CONTENTS

Data computing middleware

An integratable computing layer to feed data to an app

Data preparation

Prepare data for data mining

Ad hoc computation

Handle ad hoc queries and data extraction

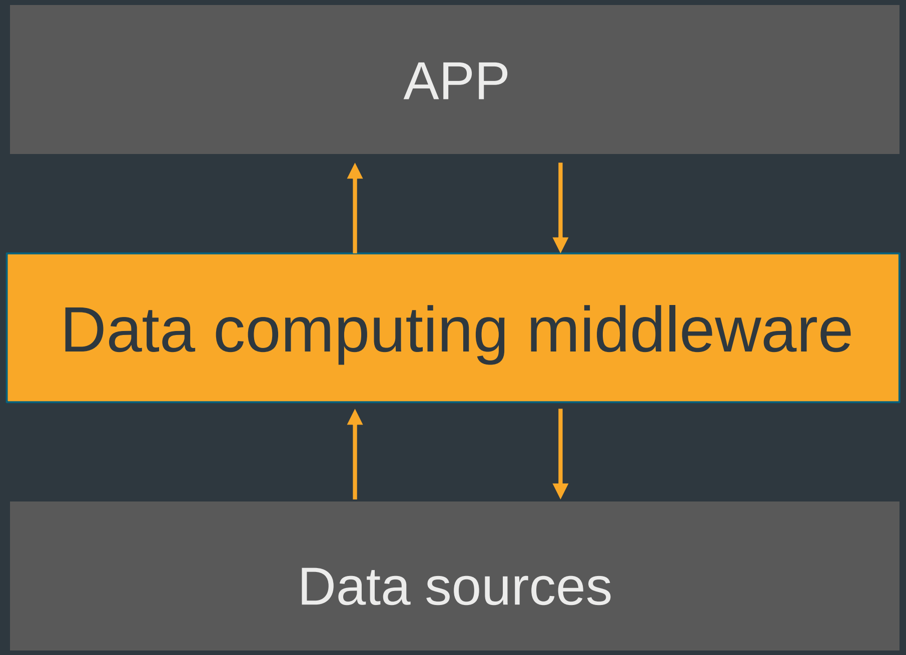
Desktop analysis

Dynamic single-machine analysis

APP

Data computing middleware

A data computing module situated between a data source and an app, a DCM offers open computing ability, shares the conventional responsibility of a data source, and reduces coupling

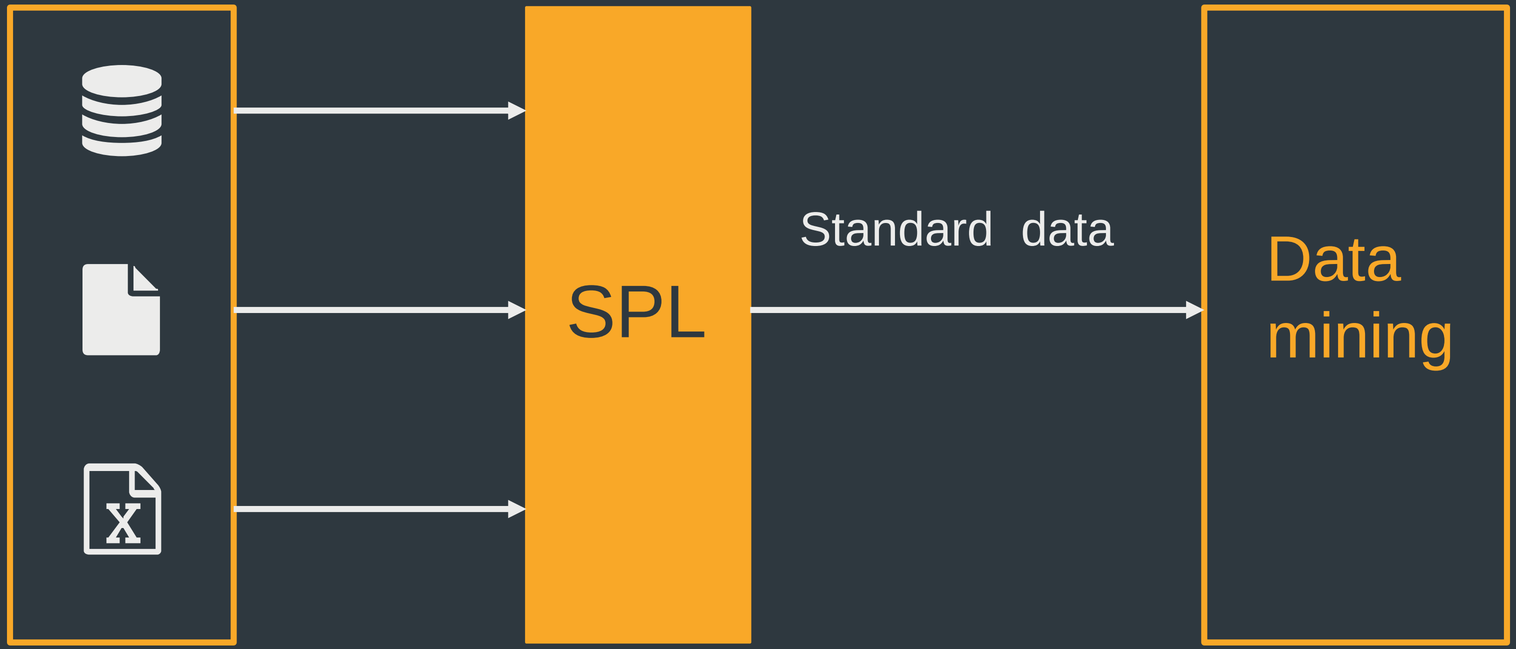


*Refer to <Data_Computing_Middleware.pptx>

Prepare data for data mining

Data preparation takes up over half of the data mining workloads;

SPL enables an open, flexible and simple method

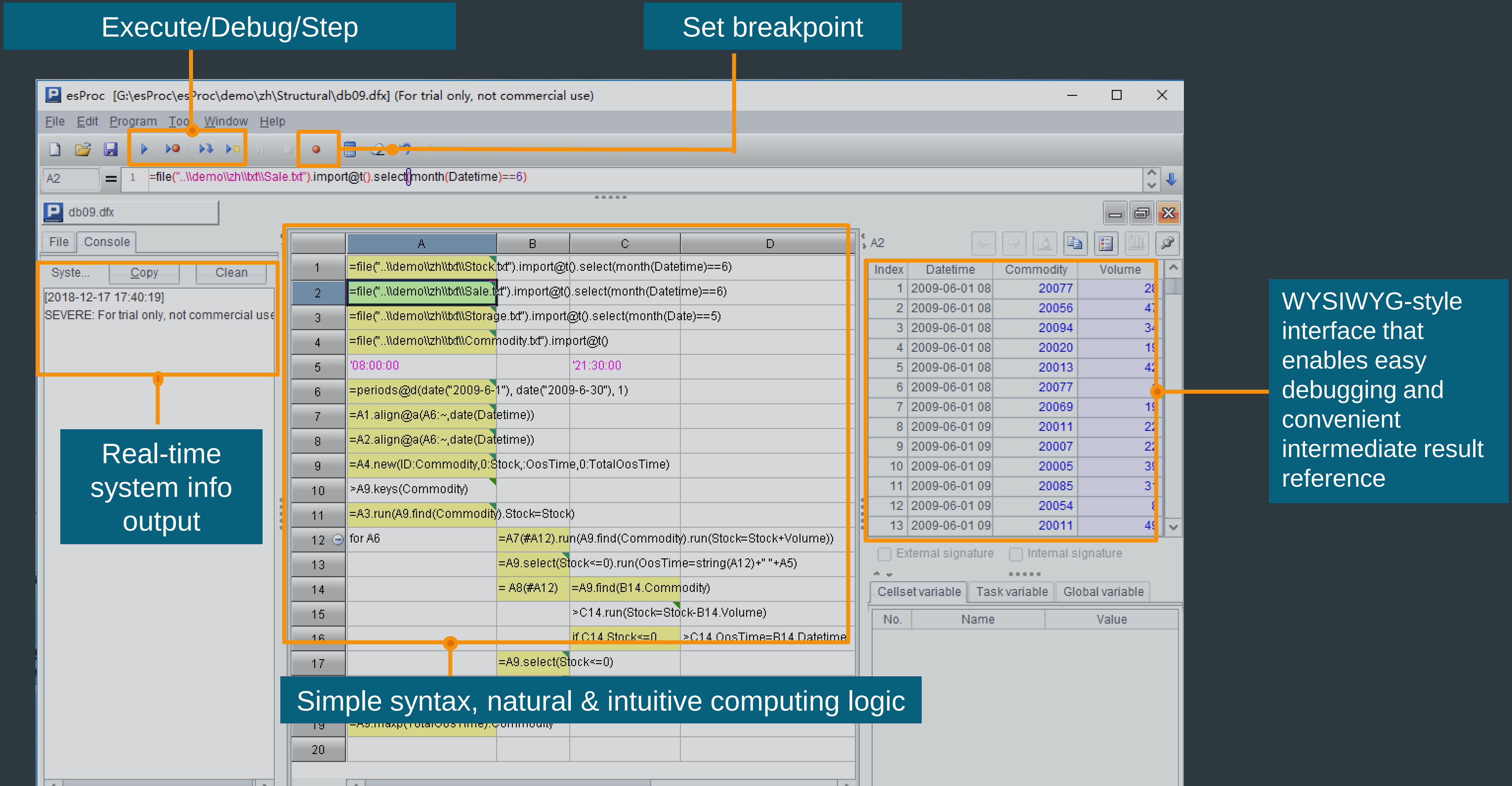


Ad hoc computation

SPL's open computing ability can:

- Handle improvised data extraction;
- Handle spontaneous (external) data analysis & research;
- Generate test data according to business rules;
- Test optimization solutions to big data processing;
- Handle unconventional (external) data cleansing & loading.

Desktop analysis



CONTENTS

Code examples

Text processing

Non-structured data

Structured data

Text-like data

Database computing

Grouping operation

Order-based computation

String & date handling

Non-structured data – Text processing

Real-world problem

Data items in each line of *T.txt* are separated by an unspecified number of spaces:

2010-8-13   991003 3166.63 3332.57 3166.63 3295.11

2010-8-10   991003 3116.31 3182.66 3084.2   3140.2

……

List average of the last data items in each line.

	A
1	=file("T.txt").read@n().(~.split@tp(“ ”).to(-4).avg())

read@n() reads the text as a set of strings; *split@t(“ ”)* splits the set into a set of subsets by the spaces; *@p* option parses each data item into a proper type automatically for calculation of averages.

Non-structured data - Structuralization

Real-world problem

In log file *S.log*, every 3 lines constitutes a piece of information.

Parse the file into structured data and save it to *T.txt*.

	A	B	
1	=file("S.log").read@n()		
2	=create(…)		Create the target result set
3	for A1.group((#-1)\3)	…	Group the file every 3 lines
…		…	Parse field values from A3's 3 lines
…		>A2.insert(…)	Insert values into the result set
…	>file("T.txt").export(A2)		Export the parsed data

With "**group by line number**" mechanism, we can handle groups one by one by loop, which is easier. The special case is that there is only one line in each group.

Non-structured data – Data searching

Real-world problem

Not all OS support the grep command; and it's not easy to realize it with code.

There are multiple text files in a directory. Find every file containing the specified word and list the line(s) holding the word and its(their) number(s).

	A
1	=directory@p("*.txt")
2	=A1.conj(file(~).read@n().(if(pos(~,"xxx"),[A1.~,#,~].string())).select(~))

With the abilities of **file traversal** and text processing, esProc can get it done with two lines of code.

Structured data – Read & Write

Real-world problem

Comma-separated *D.csv* has multiple columns, each of which has a title.

Read in 4 columns: name,sex,age,phone; read numeric column phone as string type

	A
1	=file("D.csv").import@tc(name,sex,age,phone:string)

import() function has rich parameters and options to determine if titles are read in or written out, which delimiter is used, which columns will be read/write in and which data types they will be. Most of the structured text can be read/write in with a one-liner.

This is similar to reading in a database table.

Structured data – Regular queries

Real-world problem

Find from text file *D.csv* men who are 25 and above and women who are 23 and above, and 1) List them in alphabetical order of names； 2) Group them by gender and calculate age averages； 3) List all surnames.

	A	
1	=file(“D.csv”).import@tc(name,sex,age)	
2	=A1.select(sex== “男”&& age>=25||sex== “女”&& age>=23)	Filtering
3	=A2.sort(name)	Sorting
4	=A2.groups(sex;avg(age):age)	Grouping and aggregation
5	=A2.id(left(name,1))	Find distinct values

esProc provides a rich variety of structured data computing functionalities to be able to treat the text file as, to some extent, **a database table**.

Structured data - File comparison

Real-world problem

Both text files *T1.txt* and *T2.txt* have an id column:

Find their common id values;

Find id values existing in *T1.txt* but don't exist in *T2.txt*.

	A	
1	=file("T1.txt").import@ti(id)	With @i option, return a single-column result as a seqeunce
2	=file("T2.txt").import@ti(id)	
3	=A1^A2	Intersection, which contains the common values
4	=A1\A2	Difference, which contains values that exist in T1 but don't exist in T2

Use **intersection and difference operations** to compare column values

Text-like data - JSON

Real-world problem

Java has a sufficient rich class library to parse and generate JSON data, but it lacks the ability to further compute the data;

esProc supports multilevel data. It can completely parse JSON data into a computable memory data table for further processing.

```
{
"order":[
        {
        "client":"Raqsoft" ,
        "date":"2015-6-23",
        "item" : [
                {"product":"HPLaptop" ,"number":4,"price":3200},
                {"product":"DELLSever" ,"number":1,"price":22100}]
        },…]
}
```

Write JSON data to database:

Structure of *order* table: orderid,client,date;

Structure of *orderdetail* table :

orderid,seq,product,number,price

orderid and seq values are sequently generated.

Text-like data - JSON

```
{
"order":[
        {
        "client":"Raqsoft" ,
        "date":"2015-6-23",
        "item" : [
                {"product":"HPLaptop" ,"number":4,"price":3200},

{"product":"DELLServer" ,"number":1,"price":22100}]
        },…]
}
```

	A
1	=file("data.json").read().import@j().order
2	=A1.new(#:orderid,client,date)
3	=A1.news(item;A1.#:orderid,#:seq,product,number,price)
4	>db.update@i(A2,order)
5	>db.update@i(A3,ordedetail)

Text-like data - Excel

Real-world problem

An Excel file is a structured file. Java's powerful yet low-level open-source class libraries (like poi) can parse xls files, but the development process is complex;

esProc encapsulates poi to read in an xls file as a 2-dimensional data table for further processing.

- *position.xls* stores positions of points; *range.xls* stores start points and end points of ranges.
- For each point in *position.xls*, find the first range from *range.xls* containing this point;
- Write points and their corresponding ranges to *result.xls*.

range.xls		
range	start	stop
Range1	4561	6321
Range2	9842	11253
…		

position.xls	
Point	position
point1	5213
point2	10254
…	

Text-like data - Excel

range.xls				position.xls	
range	start	stop		Point	position
Range1	4561	6321		point1	5213
Range2	9842	11253		point2	10254
…				…	

	A
1	=file(“range.xls”).importxls@t()
2	=file(“position.xls”).importxls@t()
3	=A2.derive((t=A1.select@1(position>=start&&position<=stop)).range:range,t.start:start,t.stop:stop)
4	=file(“result.xls”).exportxls(A3)

esProc makes best use of its built-in computing abilities to process an imported xls file;

Excel VBA can hard-code JOINs, but the process is complicated. Sometimes data needs to be exported to the database to be processed.

Dynamic columns – Cross-column summarization

Real-world problem

Structure of *PETestResults* table: Name, Sprint, Long-distance running, long jump, shot put…; there are 4 grade levels: Excellent, Good, Pass, and Fail. Count students of every grade level over all events.

	A	
1	=db.query("select * from PETestResults")	
2	=A1.conj(~.array().to(2,))	Concatenate sequences of students’ grades for all events (beginning from column 2)
3	=A2.groups(~:Grades;count(1):Number)	Grouping & aggregation

esProc can **get values from multiple columns to generate a sequence**, over which the processing of a dynamic number of columns becomes convenient

Dynamic columns - Transposition

Real-world problem

According to account balance table *T*, an Account Balance Report of a specified month is expected. The report will display each day's account balance in a certain month; if the balance of a date remains unchanged, record it as that of the previous day.

SQL headaches

It is a static transposition. But as the involved columns are many and the transposition is regular, it's hard to code it statically;

Cross-column computation is involved; it's not easy to code it in SQL, even using PIVOT

Account Balance Table

ID	Account	Balance	Date
1	A	Deficit	2014-1-4
2	A	Normal	2014-1-8
3	A	Missing	2014-3-21
…			

Account Balance Report

Account	1	2	3	4	5	6	7	8	9	…	31
A				Deficit	Deficit	Deficit	Deficit	Normal	Norma	…	Norma
…											

Dynamic columns - Transposition

ID	Account	Balance	Date
1	A	Deficit	2014-1-4
2	A	Normal	2014-1-8
3	A	Missing	2014-3-21
…			

Account	1	2	3	4	5	6	7	8	9	…	31
A				Deficit	Deficit	Deficit	Deficit	Normal	Normal	…	Normal
…											

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

Standard process of performing transposition: Use macro to generate the target result set in A2; transpose data by loop in A3-B6 and insert values to the result set

Data grouping – interval-based non-equi-grouping

Real-world problem

Group data by intervals, such as grade levels (excellent, good…) and age groups (young, middle-aged…).

penum() function returns sequence numbers of enum conditions: [”?<60”,”?>=60&&?<75”, ”?>=75&&?<90”, “?>=90”].penum(scores)

pseg() function easily gets sequence numbers of the intervals from a continuous array, like [60,75,90].pseg(scores)

Both enum conditions and a continuous interval are arrays that can be passed in as parameters and that are unrestricted in length;

With sequence numbers, we can convert interval-based grouping into regular equi-grouping.

Data grouping – Non-equi-grouping by specified order

Real-world problem

Group data by a specified order. For instance, put Beijing at the beginning when sorting provinces in China.

- In esProc, *align@s()* function is used to perform alignment sorting：

  T.align@s([“Beijing” ,”Hebei” ,”Shandong” ,…],Provinces)

- The code sorts province table *T* in a specified order;
- The sorting condition can be passed in as a parameter.

Data grouping – Inverse grouping

Real-world problem

Structure of *Instalment* table: ID, TotalAmount, StartDate, NumberOfInstalments;

Split each loan into records of instalments, the structure is: ID, InstalmentNumber, DueDate, MonthlyAmount. A total amount will be evenly distributed among monthly-payed instalments.

	A
1	=db.query("select * from Instalment")
2	=A1.news(NumberOfInstalments;ID,~:InstalmentNumber,after@m(StartDate,~-1):DueDate,TotalAmount/NumberOfInstalments:MonthlyAmount)

news() function calculates field values of a sequence and generate a multi-row new table sequence.

String and date handling- String

Real-world problem

A string concatenation problem. Structure of *Students* table is: Class, Name, Gender. Group the table by Class and respectively list boys and girls as comma-delimited strings sorted in alphabetical order in name.

	A
1	=db.query("select * from Students")
2	=A1.group(Class; ~.select(Gender==Male").(Name).sort().concat@c():Boys, ~.select(Gender=="Female").(Name).sort().concat@c():Girls)

esProc set data type relieves a string concatenation function of grouping operation and enables various operations

String and date handling - Date

Real-world problem

Structure of *TravelLog* table is: Name, StartDate, EndDate…;

Find the 5 peak days during the travel.

	A
1	=db.query("select StartDate,EndDate from TravelLog")
2	=A1.conj(periods(StartDate,EndDate)).groups(~:Date,count(1):Number)
3	=A2.sort(Number:-1).to(5)

It's easy to do it using the date splitting function *periods()*

THANKS

Innovation Makes Progress