QDBase

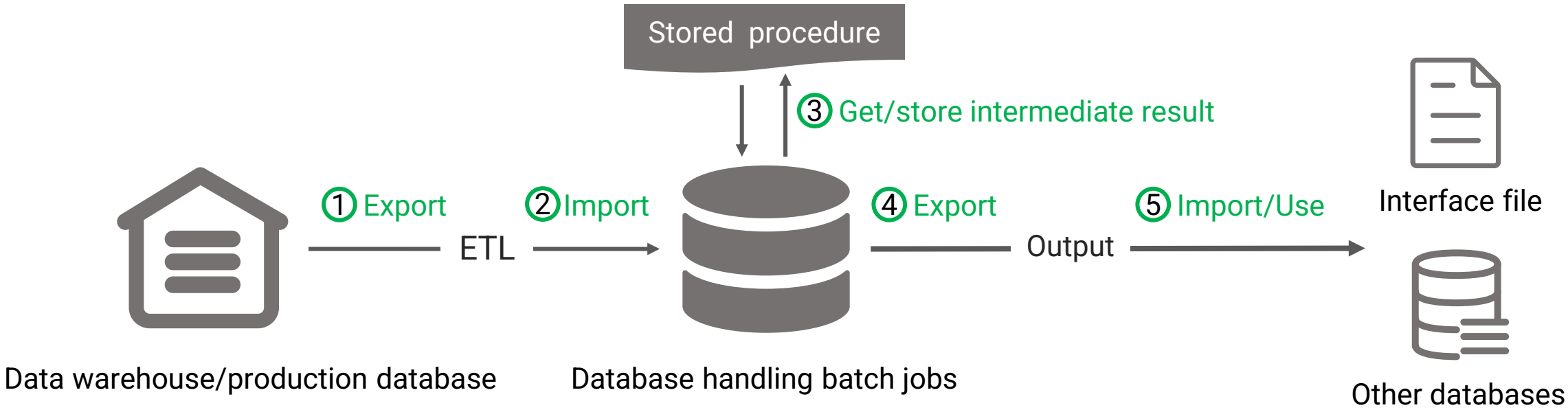
High-performance Offline Batch Job Solution and Case

Dilemmas in batch jobs

As data grows, batch jobs increase and computational load intensifies

Slow	Strained	Helpless
Job and service are delayed due to slow batch jobs	On the brink of probably prolonged delay on summary dates	Highly dependent on SP but unable to expand

Traditional way of performing batch jobs



Datawarehouse/ production database

Oracle, Teradata, Hadoop, etc.

Database handling batch jobs

Oracle, DB2, MySQL, etc.

Interface file/ Other databases

Text file/ BI platforms, business databases

Causes behind batch job problems

Extremely slow RDB read/write

Databases require to read data in for further processing due to closed storage and computing schemes

Too many verifications and processing at data read/write result in time-consuming large amounts of data import/export

Inefficient stored procedure

SQL cannot achieve high-efficiency algorithms due to syntactic deficiencies

Complex, multi-step computations involve storing intermediate results that will consume more IO resources

Database cursor has poor performance and does not support parallel processing to speed up computations

Batch jobs depend heavily on RDBs

Despite their slow running because of the latter’s uniquely sufficient computing ability

To solve those problems：

We need to put the following two capabilities in place

Openness

Compute outside-database files without I/O cost;

Access multiple/diverse data sources directly to compute mixed data；

Strong computing capacity

A rich offer of functions that can easily achieve any complex computations during batch jobs ；

High performance to ensure good efficiency;

How about using a distributed database to perform batch jobs?

Q Can we increase nodes in a distributed database to speed up batch job?

No, in many cases **we can't**! A

Poor support of stored procedure

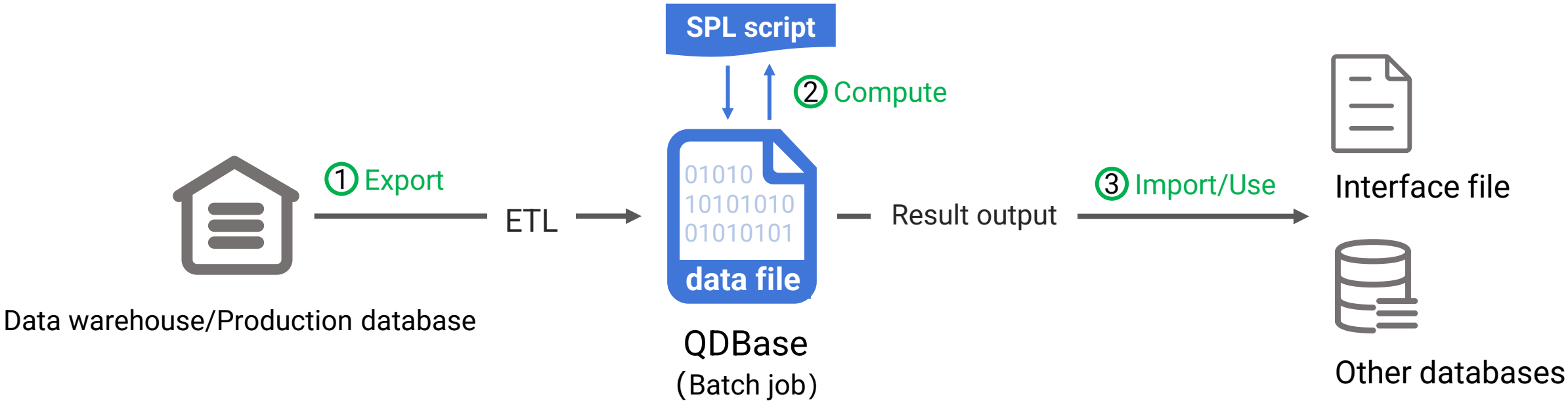
A batch job involves complex computing logic that often needs tens of thousands of lines of SP code to achieve, but a distributed DB gives poor support of SP and is under-qualified.

Heavy cross-network reads/writes caused by Intermediate result uses

The use of stored intermediate result by different nodes results in heavy cross-network reads/writes that lead to uncontrollable performance that cannot be addressed via data redundancy

Most batch jobs are thus handled in a centralized database

QDBase batch job process



Data warehouse / Production database

Oralce, Teradata, Hadoop, etc.

QDBase

Use files to store data and compute directly

Interface file / Other DBs

Text file/ BI cubes, business databases

QDBase openness – File processing

File storage advantages

Efficient storage and retrieval system

High IO performance compared to low-efficiency database storage caused by constraints and interface limitations

Flexible to use

Easy to split and store, and convenient to segment for parallel processing

Easy to administrate

Tree-structure directories are convenient to manage

QDBase file processing capabilities

Time-efficient

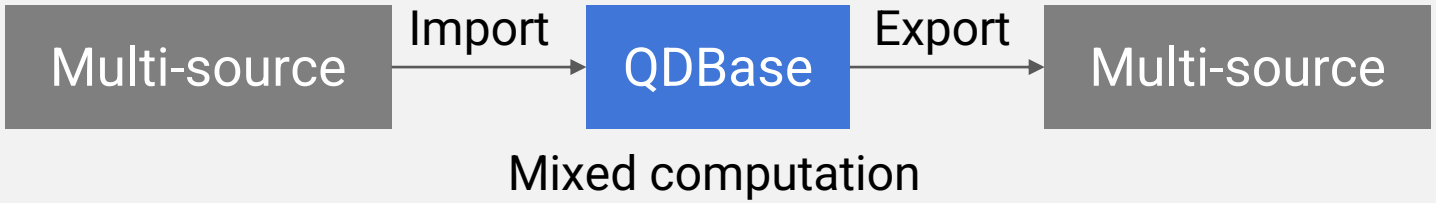
QDBase handles batch jobs directly based on data files without data import/export

High-efficiency storage

QDBase offers high-efficiency proprietary data format to further speed up batch jobs

QDBase openness – Multiple/diverse data source handling

QDBase offers multiple data source interfaces and supports efficient multi-source mixed computations that batch jobs can directly use without data import



QDBase strong computing capacity

Rich class libraries

QDBase has all-around computing capabilities with a wealth of encapsulated, ready-to-use class libraries covering from grouping, loop, sorting, filtering, set operations, order set handling to parallel processing…

Computing library

Efficient, agile syntax

QDBase has specialized, agile SPL syntax, which achieves batch job logic in a more concise way with shorter code than in SQL/SPL;

Field tests show that code amount can be times less

SPL

QDBase strong computing capacity

High performance storage

Efficient storage formats

Two proprietary storage formats – bin file and composite table that support efficient mechanisms such as compression, columnar storage and index

Order-based storage

For increase compression ratio and location performance

Double increment segmentation technique

Support any number of parallel threads

.....

High performance algorithms

On top of indispensable high-performance storage plans, there are efficient algorithms for boosting computing performance

Multi-purpose traversal

Accomplish multiple computations during one round of traverse on a large table; this helps reduce IO cost

Delayed cursor

Define multiple computing steps on one cursor to effectively reduce intermediate results to be stored

Multi-cursor

For parallel processing by making full use of multiple CPUs to increase performance

.....

QDBase Application Case

Performance Optimization of Historical Policies Association Batch Job in **Insurance Industry**

Historical policies association batch job

20,000 new policies daily, and 600,000 in one month

Task	Find historical policies corresponding to new policies

3-year historical car insurance policies:

two hundred million rows of data

Difficult points in historical policies association batch job

Complex computing rule

Three types of judgment for deciding whether it is the same car; check whether the car has loan and needs mandatory insurance

Massive data volume

Get hundreds of thousands of/millions of rows in t wo hundred million rows

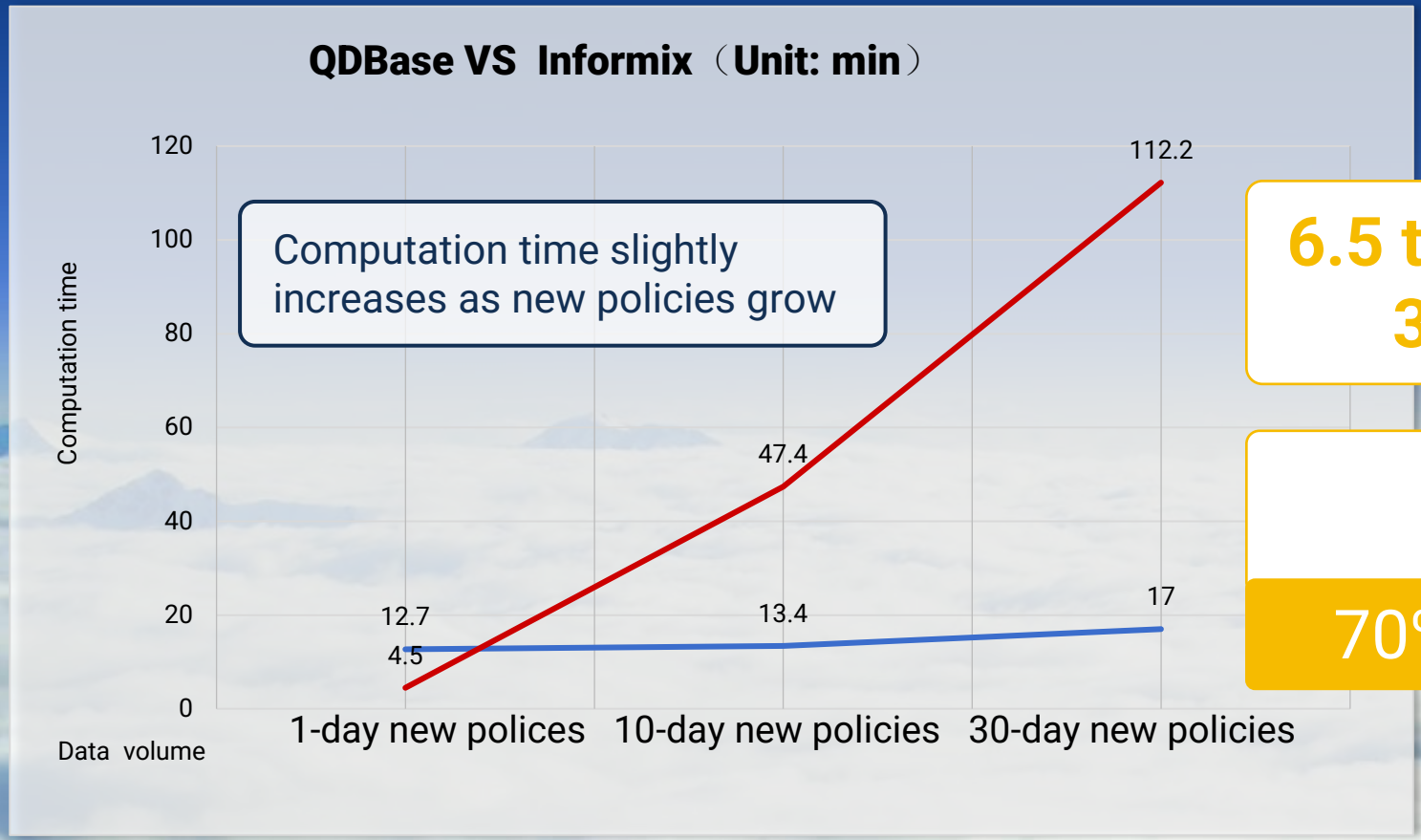
Time-consuming

2-hour for 30-day new policies
DNF for 90-day new policies

Complex computing rule

Recompute newly increased policies of the year if computing rule is changed！！

Field test: QDBase gives excellent performance



6.5 times faster to process 30-day new policies

500 cells

70% fewer code amount

Note： QDBase does not create index on historical policy table but uses traversal instead; DB creates index on it and has advantage in processing policies in a short period. As our customer's requirement only focuses on policy processing in a long period, we do not provide optimization on short-period processing scenarios.

Analysis: why QDBase is fast

Ordered data storage

Store data in order by policy number

Associate policy table and detail table by segmenting them in order

Intermediate result is ordered, and there is no need to re-create index

Data compression & columnar storage

Only a dozen among 70 fields of the policy table are involved in computation

Less than 10 of 56 fields of detail table are involved in computation

A file is 10 times smaller after compression, which effectively reduces disk reads

Analysis: QDBase offers fast algorithms

DB stored procedure

1 Associate new policies with existing policy table and detail table, and perform filtering on result table

2 Associate result1 with detail table (through VIN)

3 Filter result2 and associate policy table

…

15 Associate result1 with detail table (through chassis number)

16 Filter result15 and associate policy table

…

24 Associate result1 with detail table (through VRN)

25 Filter result24，associate policy table

…

Multiple large table associations

QDBase SPL

1. Associate policy table and detail table by segmenting them in order
2. Traverse segmentation result1 in loop
3. Increase 3 types of new policies; Associate with segmentation result1
4. Exit loop2
5. …

One segment-based association is involved

High-performance computing needs more than crafting

Many optimization algorithms and storage plans cannot be achieved in RDB (SQL), but can be accomplished in QDBase (SPL)