

**openGauss**  
**3.0.0**

# **Feature Description**

<b>Issue</b>	01
<b>Date</b>	2022-03-31



**Copyright © Huawei Technologies Co., Ltd. 2022. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **Huawei Technologies Co., Ltd.**

Address: Huawei Industrial Base  
Bantian, Longgang  
Shenzhen 518129  
People's Republic of China

Website: <https://www.huawei.com>

Email: [support@huawei.com](mailto:support@huawei.com)

---

# Contents

---

<b>1 High Performance</b>	<b>1</b>
1.1 CBO Optimizer	1
1.2 LLVM	2
1.3 Vectorized Engine	3
1.4 Hybrid Row-Column Store	4
1.5 Adaptive Compression	6
1.6 SQL by pass	8
1.7 Kunpeng NUMA Architecture Optimization	8
1.8 High Concurrency of Thread Pools	9
1.9 SMP for Parallel Execution	10
1.10 Xlog no Lock Flush	11
1.11 Parallel Page-based Redo For Ustore	12
1.12 Row-Store Execution to Vectorized Execution	13
<b>2 High Availability (HA)</b>	<b>15</b>
2.1 Primary/Standby	15
2.2 Logical Replication	17
2.3 Online Node Replacement	17
2.4 Logical Backup	18
2.5 Physical Backup	19
2.6 Automatic Job Retry upon Failure	20
2.7 Ultimate RTO	24
2.8 Cascaded Standby Server	24
2.9 Delayed Replay	26
2.10 Adding or Deleting a Standby Node	27
2.11 Delaying Entering the Maximum Availability Mode	28
2.12 Parallel Logical Decoding	29
2.13 DCF	31
2.14 CM	32
2.15 Global SysCache	33
2.16 Using a Standby Node to Build a Standby Node	35
<b>3 Maintainability</b>	<b>36</b>
3.1 Gray Upgrade	36

3.2 WDR.....	37
3.3 Slow SQL Diagnosis.....	40
3.4 Session Performance Diagnosis.....	43
3.5 System KPI-aided Diagnosis.....	45
<b>4 Database Security.....</b>	<b>47</b>
4.1 Access Control Model.....	47
4.2 Separation of Control and Access Permissions.....	48
4.3 Database Encryption Authentication.....	49
4.4 Data Encryption and Storage.....	50
4.5 Database Audit.....	51
4.6 Network Communication Security.....	52
4.7 Resource Label.....	53
4.8 Unified Audit.....	54
4.9 Dynamic Data Masking.....	57
4.10 Row-Level Access Control.....	60
4.11 Password Strength Verification.....	61
4.12 Equality Query in a Fully-encrypted Database.....	63
4.13 Ledger Database Mechanism.....	66
4.14 Transparent Data Encryption.....	68
<b>5 Enterprise-Level Features.....</b>	<b>70</b>
5.1 Support for Functions and Stored Procedures.....	70
5.2 SQL Hints.....	71
5.3 Full-Text Indexing.....	72
5.4 Copy Interface for Error Tolerance.....	73
5.5 Partitioning.....	74
5.6 Support for Advanced Analysis Functions.....	76
5.7 Materialized View.....	77
5.8 HyperLogLog.....	77
5.9 Creating an Index Online.....	78
5.10 Autonomous Transaction.....	79
5.11 Global Temporary Table.....	80
5.12 Pseudocolumn ROWNUM.....	81
5.13 Stored Procedure Debugging.....	82
5.14 JDBC Client Load Balancing and Read/Write Isolation.....	83
5.15 In-place Update Storage Engine.....	84
5.16 Publication-Subscription.....	85
5.17 Foreign Key Lock Enhancement.....	86
5.18 Data Compression in OLTP Scenarios.....	87
<b>6 Application Development Interfaces.....</b>	<b>89</b>
6.1 Standard SQL.....	89
6.2 Standard Development Interfaces.....	90

6.3 PostgreSQL API Compatibility.....	90
6.4 PL/Java.....	91
<b>7 AI Capabilities.....</b>	<b>94</b>
7.1 AI4DB: Autonomous Database O&M.....	95
7.1.1 Database Metric Collection, Forecast, and Exception Detection.....	95
7.1.2 Root Cause Analysis for Slow SQL Statements.....	96
7.1.3 Index Recommendation.....	97
7.1.4 Parameter Tuning and Diagnosis.....	98
7.1.5 Slow SQL Statement Discovery.....	99
7.2 DB4AI: Database-driven AI.....	100
7.3 AI in DB.....	101
7.3.1 Predictor: AI Query Time Forecasting.....	101
<b>8 Tool Chain.....</b>	<b>104</b>
8.1 MySQL to openGauss Migration Tool Chameleon.....	104
8.2 OpenGauss Client Tool DataStudio.....	105
<b>9 Middleware.....</b>	<b>107</b>
9.1 Distributed Database Capability.....	107
9.2 Deploying a Distributed Database Using Kubernetes.....	108

# 1 High Performance

---

- [1.1 CBO Optimizer](#)
- [1.2 LLVM](#)
- [1.3 Vectorized Engine](#)
- [1.4 Hybrid Row-Column Store](#)
- [1.5 Adaptive Compression](#)
- [1.6 SQL by pass](#)
- [1.7 Kunpeng NUMA Architecture Optimization](#)
- [1.8 High Concurrency of Thread Pools](#)
- [1.9 SMP for Parallel Execution](#)
- [1.10 Xlog no Lock Flush](#)
- [1.11 Parallel Page-based Redo For Ustore](#)
- [1.12 Row-Store Execution to Vectorized Execution](#)

## 1.1 CBO Optimizer

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

The openGauss optimizer is cost-based optimization (CBO).

### Benefits

The openGauss CBO optimizer can select the most efficient execution plan among multiple plans based on the cost to meet customer service requirements to the maximum extent.

## Description

By using CBO, the database calculates the number of tuples and the execution cost for each step under each execution plan based on the number of table tuples, column width, null record ratio, and characteristic values, such as distinct, MCV, and HB values, and certain cost calculation methods. The database then selects the execution plan that takes the lowest cost for the overall execution or for the return of the first tuple.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 1.2 LLVM

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

openGauss provides the Low Level Virtual Machine (LLVM) technology to query dynamic compilation execution.

## Benefits

The requery performance is greatly improved by dynamically building and executing queries.

## Description

Based on the query execution plan tree, with the library functions provided by the LLVM, openGauss moves the process of determining the actual execution path from the executor phase to the execution initialization phase. In this way, problems such as function calling, logic condition branch determination, and a large amount of data reading that are related to the original query execution are avoided, to improve the query performance.

## Enhancements

None.

## Constraints

None.

## Dependencies

It depends on the LLVM open-source component. Currently, the open-source version 10.0.0 is used.

# 1.3 Vectorized Engine

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

The vectorized execution engine, provided by openGauss, is usually used in OLAP data warehouse systems because analytical systems are usually data-intensive and access most data in a table in a sequential manner, perform calculation, and finally output a calculation result to an end user.

## Benefits

Batch calculation greatly improves the performance of complex query.

## Description

The traditional database query execution uses the tuple-based pipeline execution mode. In most time, the CPU is not used to actually process data, but to traverse the query operation tree. As a result, the effective utilization of the CPU is not high. This also results in low instruction cache performance and frequent jumps. Worse still, this approach does not take advantage of the new capabilities of the new hardware to speed up the execution of queries. In the execution engine, another solution is to change a tuple to a column at a time. This is also the basis of our vectorized execution engine.

The vectorized engine is bound to the column-store technology, because data of each column is stored together, and it may be considered that the data is stored in an array manner. Based on such a feature, when a same operation needs to be performed on the column data, calculation of each value of the data block may be efficiently completed by using a cycle.

The advantages of the vectorized execution engine are as follows:

- This reduces inter-node scheduling and improves CPU usage.
- Because the same type of data is put together, it is easier to leverage the new optimization features of hardware and compilation.

## Enhancements

None.



## Constraints

None.

## Dependencies

It depends on column store.

# 1.4 Hybrid Row-Column Store

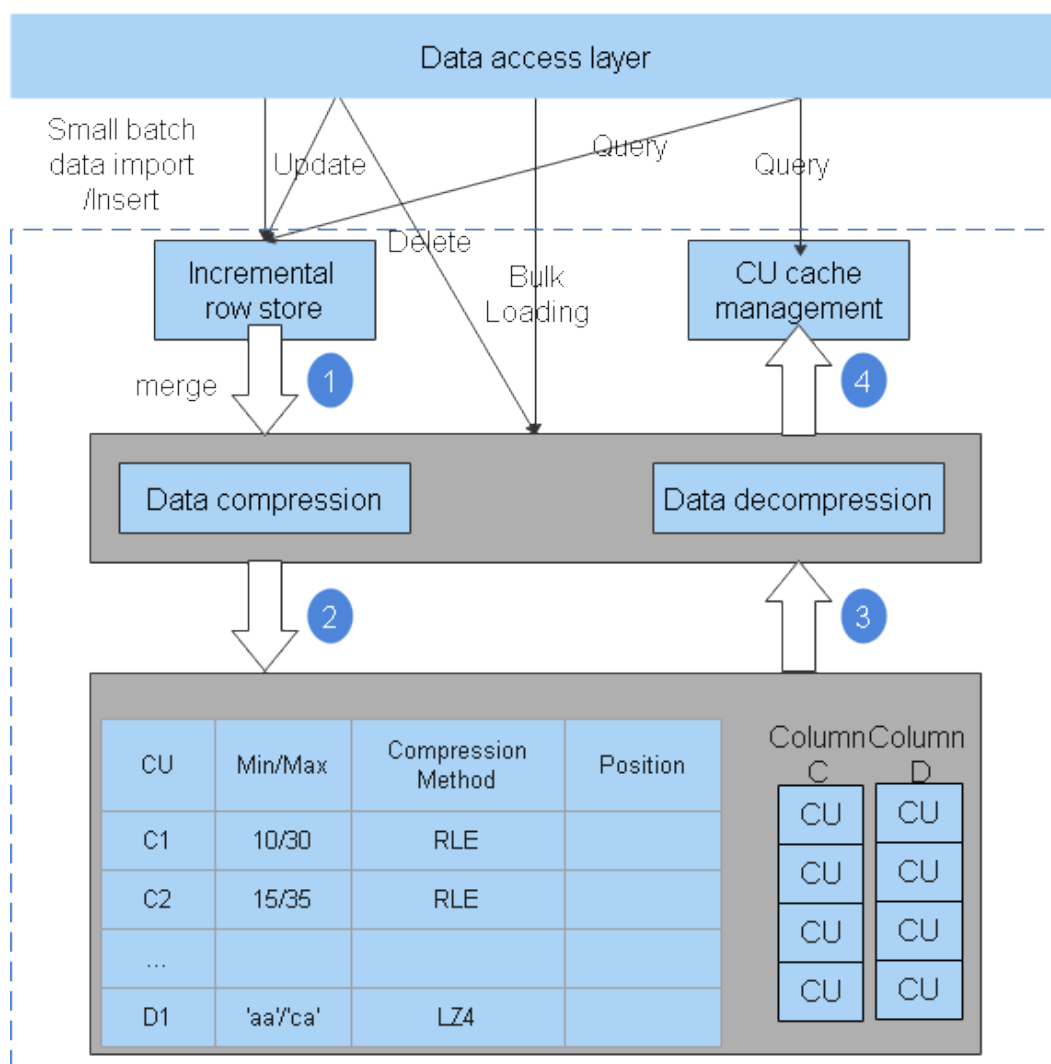
## Availability

This feature is available since openGauss 1.0.0.

## Introduction

openGauss supports both row-store and column-store models. Choose a row-store or column-store table as needed.

Column-store is recommended if a table contains many columns (called a wide table) but its query involves only a few columns. [Figure 1-1](#) shows the column-store model. Row store is recommended if a table contains only a few columns and a query involves most of the fields.

**Figure 1-1** Column-store

## Benefits

In a wide table containing a huge amount of data, a query usually only includes certain columns. In this case, the query performance of the row-store engine is poor. For example, a single table containing the data of a meteorological agency has 200 to 800 columns. Among these columns, only 10 are frequently accessed. In this case, a vectorized execution and column-store engine can significantly improve performance by saving storage space.

## Description

Tables are categorized into row-store and column-store tables. Each storage model applies to specific scenarios. Select an appropriate model when creating a table.

- **Row-store table**  
Row-store tables are created by default. Data is stored by row. Row-store supports adding, deleting, modifying, and querying data of a complete row.

Therefore, this storage model applies to scenarios where data needs to be updated frequently.

- Column-store table

Data is stored by column. The I/O of data query in a single column is small, and column-store tables occupy less storage space than row-store tables. This storage model applies to scenarios where data is inserted in batches, less updated, and queried for statistical analysis. The performance of single point query and single record insertion in a column-store table is poor.

- Selecting a storage model

- Update frequency

If data is frequently updated, use a row-store table.

- Data insertion frequency

If a small amount of data is frequently inserted each time, use a row-store table. If a large amount of data is inserted at a time, use a column-store table.

- Number of columns

If a table is to contain many columns, use a column-store table.

- Number of columns to be queried

If only a small number of columns (less than 50% of the total) is queried each time, use a column-store table.

- Compression ratio

The compression ratio of a column-store table is higher than that of a row-store table. High compression ratio consumes more CPU resources.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 1.5 Adaptive Compression

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Data compression is the major technology used in current databases. Various compression algorithms are used for different data types. If pieces of data of the same type have different characteristics, their compression algorithms and results will also be different. Adaptive compression chooses the suitable compression

algorithm for data based on the data type and characteristics, achieving high performance in compression ratio, import, and query.

## Benefits

Importing and frequently querying a huge amount of data are the main application scenarios. When you import data, adaptive compression greatly reduces the data volume, increases I/O operation efficiency several times, and clusters data before storage, achieving fast data import. In this way, only a small number of I/O operations is required and data is quickly decompressed in a query. Data can be quickly retrieved and the query result is quickly returned.

## Description

Currently, the database has implemented various compression algorithms on column store, including RLE, DELTA, BYTEPACK/BITPACK, LZ4, ZLIB, and LOCAL DICTIONARY. The following table lists data types and the compression algorithms suitable for them.

-	RLE	DELT A	BITPACK/ BYTEPACK	LZ4	ZLIB	LOCAL DICTION ARY
Smallint/Int/Bigint/Oid Decimal/Real/Double Money/Time/Date/ Timestamp	√	√	√	√	√	-
Tinterval/Interval/Time with time zone/	-	-	-	-	√	-
Numeric/Char/Varchar/ Text/Nvarchar2 and other supported data types	√	√	√	√	√	√

## Enhancements

The compression level of compression algorithms can be adjusted.

## Constraints

None

## Dependencies

It depends on LZ4 or ZLIB.

## 1.6 SQL by pass

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

Query performance is improved by customizing an execution scheme for typical queries in the TP scenario.

### Benefits

The TP query performance is improved.

### Description

In a typical OLTP scenario, simple queries account for a large proportion. This type of queries involves only single tables and simple expressions. To accelerate such query, the SQL bypass framework is proposed. After simple mode judgment is performed on such query at the parse layer, the query enters a special execution path and skips the classic execution framework, including operator initialization and execution, expression, and projection. Instead, it directly rewrites a set of simple execution paths and directly invokes storage interfaces, greatly accelerating the execution of simple queries.

### Enhancements

None.

### Constraints

None.

### Dependencies

None.

## 1.7 Kunpeng NUMA Architecture Optimization

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

Kunpeng NUMA architecture optimization mainly focuses on Kunpeng processor architecture features and ARMv8 instruction set, and optimizes the system from multiple layers, including OS, software architecture, lock concurrency, logs, atomic

operations, and cache access. This greatly improves the openGauss performance on the Kunpeng platform.

## Benefits

Transactions per minute (TPM) is a key performance indicator of the database competitiveness. Under the same hardware costs, a higher database performance means the database can process more services, thereby reducing the usage cost of customers.

## Description

- openGauss optimizes the Kunpeng NUMA architecture based on the architecture characteristics. This reduces cross-core memory access latency and maximizes multi-core Kunpeng computing capabilities. The key technologies include redo log batch insertion, NUMA distribution of hotspot data, and CLog partitions, greatly improving the TP system performance.
- Based on the ARMv8.1 architecture used by the Kunpeng chip, openGauss uses the LSE instruction set to implement efficient atomic operations, effectively improving the CPU usage, multi-thread synchronization performance, and Xlog write performance.
- Based on the wider L3 cache line provided by the Kunpeng chip, openGauss optimizes hotspot data access, effectively improving the cache access hit ratio, reducing the cache consistency maintenance overhead, and greatly improving the overall data access performance of the system.
- Kunpeng 920, 2P server (64 cores x 2, memory: 768 GB), 10 GE network, I/O: 4 NVMe PCIe SSDs, TPC-C: 1000 warehouses, performance: 1,500,000 tpmC.

## Enhancements

- Batch redo log insertion and CLog partition are supported, improving the database performance on the Kunpeng platform.
- Efficient atomic operations using the LSE instruction set are supported, improving multi-thread synchronization performance.

## Constraints

None

## Dependencies

None

# 1.8 High Concurrency of Thread Pools

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

The thread pooling technology is used to support stable running of databases at high concurrency.

## Benefits

The overall system throughput is stable in case of a large number of concurrent requests.

## Description

The overall design idea of the thread pool technology is to pool thread resources and reuse them among different connections. After the system is started, a fixed number of working threads are started based on the current number of cores or user configuration. A working thread serves one or more connection sessions. In this way, the session and thread are decoupled. The number of worker threads is fixed. Therefore, frequent thread switchover does not occur in case of high concurrency. The database layer schedules and manages sessions.

## Enhancements

This feature is available since openGauss 1.0.0.

In openGauss 1.1.0, thread pools can be dynamically scaled in or out.

## Constraints

None

## Dependencies

None

# 1.9 SMP for Parallel Execution

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

The Symmetric Multi-Processing (SMP) technology of openGauss uses the multi-core CPU architecture of a computer to implement multi-thread parallel computing, fully using CPU resources to improve query performance.

## Benefits

Fully utilizes the system multi-core capability to improve requery performance.

## Description

In complex query scenarios, a single query takes long time and the system concurrency is low. Therefore, the SMP technology is used to implement operator-level parallel execution, which effectively reduces the query time and improves the query performance and resource utilization. The overall implementation of the SMP technology is as follows: For query operators that can be executed in parallel, data is sliced, multiple working threads are started for computation, and then the results are summarized and returned to the frontend. The data interaction operator **Stream** is added to the SMP architecture to implement data interaction between multiple working threads, ensuring the correctness and integrity of the query.

## Enhancements

None.

## Constraints

- Index scanning cannot be executed in parallel.
- MergeJoin cannot be executed in parallel.
- WindowAgg order by cannot be executed in parallel.
- The cursor cannot be executed in parallel.
- Queries in stored procedures and functions cannot be executed in parallel.
- Subplans and initplans cannot be queried in parallel, and operators that contain subqueries cannot be executed in parallel, either.
- Query statements that contain the median operation cannot be executed in parallel.
- Queries with global temporary tables cannot be executed in parallel.
- Updating materialized views cannot be executed in parallel.

## Dependencies

None.

# 1.10 Xlog no Lock Flush

## Availability

This feature is available since openGauss 2.0.0.

## Introduction

Canceled the WalInsertLock contention and dedicated WalWriter disk write thread.

## Benefits

The system performance is further improved on the basis that the original Xlog functions remain unchanged.



## Description

This feature optimizes the WalInsertLock mechanism by using log sequence numbers (LSNs) and log record counts (LRCs) to record the copy progress of each backend. The backend can directly copy logs to the WalBuffer without contending for the WalInsertLock. In addition, a dedicated WALWriter thread is used to write logs, and the backend thread does not need to ensure the Xlog flushing.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 1.11 Parallel Page-based Redo For Ustore

## Availability

This feature is available since openGauss 2.1.0.

## Introduction

Optimized Ustore inplace update WAL write and improved the degree of parallelism for Ustore DML operation replay.

## Benefits

The WAL space used by the update operation is reduced, and the degree of parallelism for Ustore DML operation replay is improved.

## Description

Prefixes and suffixes are used to reduce the write times of WAL update. Replay threads are classified to solve the problem that most Ustore DML WALs are replayed on multiple pages. In addition, Ustore data pages are replayed based on **blkno**.

## Enhancements

None.

## Constraints

None.

## Dependencies

This feature depends on the Ustore engine.

# 1.12 Row-Store Execution to Vectorized Execution

## Availability

This feature is available since openGauss 3.0.0.

## Introduction

Row-store table queries are converted into vectorized execution plans for execution, improving the execution performance of complex queries.

## Benefits

When processing row-store table queries, the executor has poor performance when executing complex queries that include many expressions or join operations, while the vectorized executor has excellent performance when executing complex queries. Therefore, query performance of complex queries can be effectively improved by converting row-store table execution into vectorized execution.

## Description

This feature adds a RowToVec operation to scan operators. After the data in the row-store table is converted into the vectorized format in the memory, the upper-layer operators can be converted into vectorized operators. In this way, the vectorized executor can be used for calculation. Scan operators that support this conversion include SeqScan, IndexOnlyScan, IndexScan, BitmapScan, FunctionScan, ValueScan and TidScan.

## Enhancements

None.

## Constraints

- Vectorization is not supported in the following scenarios:
  - The targetList contains a function that returns set.
  - The targetList or qual contains expressions that do not support vectorization, such as array expressions, multi-subquery expressions, field expressions, and system catalog columns.
  - The following types do not support vectorization: POINTOID, LSEG OID, BOXOID, LINEOID, CIRCLEOID, POLYGONOID, PATHOID, and user-defined types.
- MOTs do not support vectorization.

## Dependencies

Vectorized executor

# 2 High Availability (HA)

---

- [2.1 Primary/Standby](#)
- [2.2 Logical Replication](#)
- [2.3 Online Node Replacement](#)
- [2.4 Logical Backup](#)
- [2.5 Physical Backup](#)
- [2.6 Automatic Job Retry upon Failure](#)
- [2.7 Ultimate RTO](#)
- [2.8 Cascaded Standby Server](#)
- [2.9 Delayed Replay](#)
- [2.10 Adding or Deleting a Standby Node](#)
- [2.11 Delaying Entering the Maximum Availability Mode](#)
- [2.12 Parallel Logical Decoding](#)
- [2.13 DCF](#)
- [2.14 CM](#)
- [2.15 Global SysCache](#)
- [2.16 Using a Standby Node to Build a Standby Node](#)

## 2.1 Primary/Standby

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

To ensure that a fault can be rectified, data needs to be written into multiple copies. Multiple copies are configured for the primary and standby nodes, and logs

are used for data synchronization. In this way, openGauss has no data lost when a node is faulty or the system restarts after a stop, meeting the ACID feature requirements.

## Benefits

Services can be switched to the standby node when the primary node is faulty. Therefore, data is not lost and services can be quickly restored.

## Description

You can deploy the one-primary-multiple-standby mode. In the one-primary-multiple-standby mode, all standby nodes need to redo logs and can be promoted to the primary. The one-primary-multiple-standby mode provides higher DR capabilities and is more suitable for the OLTP system that processes a large number of transactions.

The **switchover** command can be used to trigger a switchover between the primary and standby nodes. If the primary node is faulty, the **failover** command can be used to promote the standby node to the primary.

In scenarios such as initial installation or backup and restoration, data on the standby node needs to be rebuilt based on the primary node. In this case, the build function is required to send the data and WALs of the primary node to the standby node. When the primary node is faulty and joins again as a standby node, the build function needs to be used to synchronize data and WALs with those of the new primary node. In addition, in online capacity expansion scenarios, you need to use build to synchronize metadata to instances on new nodes. Build includes full build and incremental build. Full build depends on primary node data for rebuild. The amount of data to be copied is large and the time required is long. Incremental build copies only differential files. The amount of data to be copied is small and the time required is short. Generally, the incremental build is preferred for fault recovery. If the incremental build fails, the full build continues until the fault is rectified.

To implement HA DR for all instances, in addition to the preceding primary/standby multi-copy replication configured for DNs, openGauss also provides other primary/standby DR capabilities, such as CM server (one primary and multiple standbys) and ETCD (one primary and multiple standbys). In this way, instances can be recovered as soon as possible without interrupting services, minimizing the impact of hardware, software, and human errors on services and ensuring service continuity.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

## 2.2 Logical Replication

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

openGauss provides the logical decoding function to reversely parse physical logs to logical logs. Logical replication tools such as DRS convert logical logs to SQL statements and replay the SQL statements in the peer database. In this way, data can be synchronized between heterogeneous databases. Currently, unidirectional and bidirectional logical replication between the openGauss database and the MySQL or Oracle database is supported.

### Benefits

Logical replication is applicable to real-time database data migration, dual-database active-active system, and rolling upgrades.

### Description

DNs reversely parse physical logs to logical logs. Logical replication tools such as DRS extract logical logs from DNs, convert the logs to SQL statements, and replay the SQL statements in MySQL. Logical replication tools also extract logical logs from a MySQL database, reversely parse the logs to SQL statements, and replay the SQL statements in openGauss. In this way, data can be synchronized between heterogeneous databases.

### Enhancements

- openGauss 1.0.0 logic decoding supports the extraction of logs from full and incremental logs.
- openGauss 1.1.0 supports logical decoding on a standby node.

### Constraints

Column-store replication and DDL replication are not supported.

### Dependencies

It depends on logical replication tools that decode logical logs.

## 2.3 Online Node Replacement

### Availability

This feature is available since openGauss 1.0.0.

## Introduction

If a node in a database is unavailable or the instance status is abnormal due to a hardware fault and the database is not locked, you can replace the node or rectify the instance fault to restore the database. During the restoration, DML operations are supported. DDL operations are supported in limited scenarios only.

## Benefits

Currently, the scale of enterprise data is increasing, the number of nodes increases sharply, and the probability of hardware damage increases accordingly. The traditional offline node replacement mode cannot meet customer requirements for uninterrupted services. During routine O&M, frequent service interruption will bring great loss to customers. However, the current database products in the industry cannot meet the requirements of physical node replacement in large-scale data scenarios without service interruption. Services need to be interrupted, or only some operations are allowed when services are not interrupted.

## Description

If a node in a database is unavailable or the instance status is abnormal due to a hardware fault, you can replace the node or rectify the instance fault to restore the database. During the restoration, DML operations are supported. DDL operations are supported in limited scenarios only.

## Enhancements

None.

## Constraints

Currently, online DDL operations are supported during node replacement.

- During node replacement, DML operations are supported and DDL operations are supported in certain scenarios.

## Dependencies

None.

# 2.4 Logical Backup

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Data in user tables in the database is backed up to a specified storage medium in a general format.

## Benefits

Through logical backup, you can achieve the following purposes:

- Back up user data to a reliable storage medium to secure data.
- Support cross-version recovery and heterogeneous recovery using a general data format.
- Archive cold data.

## Description

openGauss provides the logical backup capability to back up data in user tables to local disk files in text or CSV format and restore the data in homogeneous or heterogeneous databases.

## Enhancements

None.

## Constraints

For details about the restrictions on logical backup, see "Server Tools > gs\_dump" in the *Tool Reference*.

## Dependencies

None.

# 2.5 Physical Backup

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Data in the entire database is backed up to a specified storage medium in an internal format.

## Benefits

Through physical backup, you can achieve the following purposes:

- Back up data of the entire database to a reliable storage medium, improving system reliability.
- Improve backup and restoration performance using an internal data format.
- Archive cold data.

The typical physical backup policy and application scenario are as follows:

- On Monday, perform a full backup of the database.



- On Tuesday, perform an incremental backup based on the full backup on Monday.
- On Wednesday, perform an incremental backup based on the incremental backup on Tuesday.
- ...
- On Sunday, perform an incremental backup based on the incremental backup on Saturday.

The preceding backup operations are executed every week.

## Description

openGauss 1.1.0 provides the physical backup capability to back up data of the entire database to local disk files, OBS objects, NBU objects, or EISOO objects in the internal database format, and restore data of the entire database in a homogeneous database. In addition to the preceding functions, it also provides advanced functions such as compression, flow control, and resumable backup.

Physical backup is classified into full backup and incremental backup. The difference is as follows: Full backup includes the full data of the database at the backup time point. The time required for full backup is long (in direct proportion to the total data volume of the database), and a complete database can be restored. An incremental backup involves only incremental data modified after a specified time point. It takes a short period of time (in direct proportion to the incremental data volume and irrelevant to the total data volume). However, a complete database can be restored only after the incremental backup and full backup are performed.

## Enhancements

Supports full backup and incremental backup simultaneously.

## Constraints

For details about the constraints of physical backup, see Backup and Restoration > Physical Backup and Restoration in *Administrator Guide*.

## Dependencies

None.

# 2.6 Automatic Job Retry upon Failure

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

If an error occurs in batch processing jobs due to network exceptions or deadlocks, failed jobs are automatically retried.

## Benefits

In common fault scenarios, such as network exception and deadlock, queries retry automatically in case of failure to improve database usability.

## Description

openGauss provides the job retry mechanism: gsql Retry.

- The gsql retry mechanism uses a unique error code (SQL STATE) to identify an error that requires a retry. The function of the client tool gsql is enhanced. The error code configuration file **retry\_errcodes.conf** is used to configure the list of errors that require a retry. The file is stored in the installation directory at the same level as gsql. **gsql** provides the **\set RETRY [number]** command to enable or disable the retry function. The number of retry times ranges from 5 to 10, and the default value is 5. When this function is enabled, **gsql** reads the preceding configuration file. The error retry controller records the error code list through the container. If an error occurs in the configuration file after the function is enabled, the controller sends the cached query statement to the server for retry until the query is successful or an error is reported when the number of retry times exceeds the maximum.

## Enhancements

None

## Constraints

- Functionality constraints:  
Retrying increases execution success rate but does not guarantee success.
- Error type constraints:  
Only the error types in [Table 2-1](#) are supported.

**Table 2-1** Supported error types

Error Type	Error Code	Remarks
CONNECTION_RESET_BY_PEER	YY001	TCP communication error. Print information: "Connection reset by peer"
STREAM_CONNECTION_RESET_BY_PEER	YY002	TCP communication error. Print information: "Stream connection reset by peer" (communication between DNs)
LOCK_WAIT_TIMEOUT	YY003	Lock wait timeout. Print information: "Lock wait timeout"

Error Type	Error Code	Remarks
CONNECTION_TIMED_OUT	YY004	TCP communication error. Print information: "Connection timed out"
SET_QUERY_ERROR	YY005	Failed to deliver the <b>SET</b> command. Print information: "Set query error"
OUT_OF_LOGICAL_MEMORY	YY006	Failed to apply for memory. Print information: "Out of logical memory"
SCTP_MEMORY_ALLOC	YY007	SCTP communication error. Print information: "Memory allocate error"
SCTP_NO_DATA_IN_BUFFER	YY008	SCTP communication error. Print information: "SCTP no data in buffer"
SCTP_RELEASE_MEMORY_CLOSE	YY009	SCTP communication error. Print information: "Release memory close"
SCTP_TCP_DISCONNECT	YY010	SCTP and TCP communication error. Print information: "SCTP, TCP disconnect"
SCTP_DISCONNECT	YY011	SCTP communication error. Print information: "SCTP disconnect"
SCTP_REMOTE_CLOSE	YY012	SCTP communication error. Print information: "Stream closed by remote"
SCTP_WAIT_POLL_UNKNOW	YY013	Waiting for an unknown poll. Print information: "SCTP wait poll unknow"
SNAPSHOT_INVALID	YY014	Invalid snapshot. Print information: "Snapshot invalid"
ERRCODE_CONNECTION_RECEIVE_WRONG	YY015	Failed to receive a connection. Print information: "Connection receive wrong"
OUT_OF_MEMORY	53200	Out of memory. Print information: "Out of memory"

Error Type	Error Code	Remarks
CONNECTION_EXCEPTION	08000	Failed to communicate with DNs due to connection errors. Print information: "Connection exception"
ADMIN_SHUTDOWN	57P01	System shutdown by the administrator. Print information: "Admin shutdown"
STREAM_REMOTE_CLOSE_SOCKET	XX003	Remote socket disabled. Print information: "Stream remote close socket"
ERRCODE_STREAM_DUPLICATE_QUERY_ID	XX009	Duplicate query. Print information: "Duplicate query id"
ERRCODE_STREAM_CONCURRENT_UPDATE	YY016	Concurrent stream query and update. Print information: "Stream concurrent update"

- Statement type constraints:  
Support single-statement stored procedures, functions, and anonymous blocks. Statements in transaction blocks are not supported.
- Statement constraints of a stored procedure:
  - If an error occurs during the execution of a stored procedure containing EXCEPTION (including statement block execution and statement execution in EXCEPTION), the stored procedure can be retried. If the error is captured by EXCEPTION, the stored procedure cannot be retried.
  - Advanced packages that use global variables are not supported.
  - DBE\_TASK is not supported.
  - PKG\_UTIL file operation is not supported.
- Data import constraints:
  - The **COPY FROM STDIN** statement is not supported.
  - The **gsql \copy from** metacommand is not supported.
  - Data cannot be imported using **JDBC CopyManager copyIn**.

## Dependencies

Valid only if the **gsql** tool works normally and the error list is correctly configured.

## 2.7 Ultimate RTO

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

- The database host can be quickly restored after being restarted.
- Logs can be synchronized between the primary and standby nodes to accelerate playback on the standby node.

### Benefits

When the service load is heavy, the playback speed of the standby node cannot catch up with that of the primary node. After the system runs for a long time, logs are accumulated on the standby node. If a host is faulty, data restoration takes a long time and the database is unavailable, which severely affects system availability.

The ultimate recovery time object (RTO) is enabled to reduce the data recovery time after a host fault occurs and improve availability.

### Description

After the ultimate RTO function is enabled, multi-level pipelines are established for Xlog log playback to improve the concurrency and log playback speed.

### Enhancements

None.

### Constraints

The ultimate RTO focuses only on whether the RTO of the standby node meets the requirements. The ultimate RTO removes the built-in flow control and uses the **recovery\_time\_target** parameter for flow control. This feature does not support the read operation on the standby node. If you query the standby node, a core dump may occur on the standby node.

### Dependencies

None.

## 2.8 Cascaded Standby Server

### Availability

This feature is available since openGauss 1.1.0.

## Introduction

A cascaded standby server can be connected to a standby server based on the one-primary-multiple-standby architecture.

## Benefits

The one-primary-multiple-standby architecture cannot support a flexible structure in feature service scenarios. The multi-equipment room deployment cannot meet requirements of the complete structure in the HA switchover scenario (three instances in the primary and standby equipment rooms and two or three instances in the secondary equipment room). If the number of standby servers increases, the primary server may be overloaded. Queries that have low real-time requirements can be implemented on cascaded standby servers. Therefore, the cascading backup capability is required.

## Description

The primary server replicates logs to the standby server in synchronous or asynchronous mode. The standby server replicates logs to the cascaded standby server only in asynchronous mode.

In the current one-primary-multiple-standby architecture, the primary server uses the WAL sender process (walsender) to replicate logs to the standby server. The standby server uses the WAL receiver process (walreceiver) to receive and then flushes logs to local disks. The standby server reads redo logs to complete data replication between the primary and standby servers. There is a one-to-one mapping between walsender and walreceiver on the primary and standby servers. Logs are sent between the standby and cascaded standby servers in asynchronous mode using walsender and walreceiver, reducing the streaming replication pressure on the primary server.

## Enhancements

None

## Constraints

- A cascaded standby server can only replicate data from a standby server and cannot directly replicate data from the primary server.
- A cascaded standby server does not support data build from a standby server. Currently, data can be built only from the primary server. If the standby server is fully built, the cascaded standby server needs to be fully built.
- The cascaded standby node is in asynchronous replication mode.
- The cascaded standby server cannot be promoted.
- The cascaded standby server cannot be notified.
- Currently, the overall architecture of the primary-standby-cascaded standby cluster cannot be queried. You need to find the standby server based on the primary server and then find the cascaded standby server based on the standby server.
- A cascaded standby server cannot own another cascaded standby server.

- When the ultimate RTO is enabled, no cascaded standby server is supported.

## Dependencies

None

## 2.9 Delayed Replay

### Availability

This feature is available since openGauss 2.0.0.

### Introduction

The time for a standby node to replay can be delayed.

### Benefits

By default, the standby server restores the Xlog records from the primary server as soon as possible. This function allows you to delay the time for a standby node to replay Xlog records. In this case, you can query a copy that records data before a period of time, which helps correct errors such as misoperations.

### Description

The GUC parameter **recovery\_min\_apply\_delay** can be used to set the delay time so that a standby server can replay Xlog records from the primary server after a delay time.

Value range: an integer ranging from 0 to INT\_MAX. The unit is ms.

Default value: **0** (no delay)

### Enhancements

None.

### Constraints

- The **recovery\_min\_apply\_delay** parameter is invalid on the primary node. It must be set on the standby node to be delayed.
- The delay time is calculated based on the timestamp of transaction commit on the primary server and the current time on the standby server. Therefore, ensure that the clocks of the primary and standby servers are the same.
- Operations without transactions are not delayed.
- After the primary/standby switchover, if the original primary server needs to be delayed, you need to manually set this parameter.
- When **synchronous\_commit** is set to **remote\_apply**, synchronous replication is affected by the delay. Each commit message is returned only after the replay on the standby server is complete.
- Using this feature also delays **hot\_standby\_feedback**, which may cause the primary server to bloat, so be careful when using both.

- If a DDL operation (such as DROP or TRUNCATE) that holds an AccessExclusive lock is performed on the primary server, the query operation on the operation object on the standby server will be returned only after the lock is released during the delayed replay of the record on the standby server.

## Dependencies

None.

## 2.10 Adding or Deleting a Standby Node

### Availability

This feature is available since openGauss 2.0.0.

### Introduction

Standby nodes can be added and deleted.

### Benefits

If the read pressure of the primary node is high or you want to improve the disaster recovery capability of the database, you need to add a standby node. If some standby nodes in a cluster are faulty and cannot be recovered within a short period of time, you can delete the faulty nodes to ensure that the cluster is running properly.

### Description

openGauss can be scaled out from a single node or one primary and multiple standbys to one primary and eight standbys. Cascaded standby nodes can be added. Standby nodes can be added when a faulty standby node exists in the cluster. One primary and multiple standbys can be scaled in to a single node. A faulty standby node can be deleted.

Standby nodes can be added or deleted online without affecting the primary node.

### Enhancements

None.

### Constraints

For adding a standby node:

- Ensure that the openGauss image package exists on the primary node.
- Ensure that the same users and user groups as those on the primary node have been created on the new standby node.
- Ensure that the mutual trust of user **root** and the database management user has been established between the existing database nodes and the new nodes.



- Ensure that the XML file has been properly configured and information about the standby node to be scaled has been added to the installed database configuration file.
- Ensure that only user **root** is authorized to run the scale-out command.
- Do not run the **gs\_dropnode** command on the primary node to delete other standby nodes at the same time.
- Ensure that the environment variables of the primary node have been imported before the scale-out command is run.
- Ensure that the operating system of the new standby node is the same as that of the primary node.
- Do not perform an primary/standby switchover or failover on other standby nodes at the same time.

For deleting a standby node:

- Delete the standby node only on the primary node.
- Do not perform an primary/standby switchover or failover on other standby nodes at the same time.
- Do not run the **gs\_expansion** command on the primary node for scale-out at the same time.
- Do not run the **gs\_dropnode** command twice at the same time.
- Before deletion, ensure that the database management user trust relationship has been established between the primary and standby nodes.
- Run this command as a database administrator.
- Before running commands, run the **source** command to import environment variables of the primary node.

## Dependencies

None.

## 2.11 Delaying Entering the Maximum Availability Mode

### Availability

This feature is available since openGauss 3.0.0.

### Introduction

The primary node can be delayed to enter the maximum availability mode.

### Benefits

When the primary node detects that the standby node exits due to network instability or other reasons and the maximum availability mode is enabled on the primary node, the primary node remains in the maximum protection mode within a specified time window. After the time window expires, the primary node enters the maximum availability mode.

This prevents the primary node from frequently switching between the maximum protection mode and maximum availability mode due to factors such as network jitter and intermittent process disconnection.

## Description

If **most\_available\_sync** is set to **on**, when synchronous standby nodes are faulty in primary/standby scenarios and the number of configured synchronous standby nodes is insufficient (for details, see the meaning of **synchronous\_standby\_name**), setting **keep\_sync\_window** will retain the maximum protection mode within the time window specified by **keep\_sync\_window**. That is, committing transactions on the primary node is blocked, delaying the primary node to enter the maximum availability mode.

If synchronous standby nodes recover from faults and the number of synchronous standby nodes meets the configuration requirements, transactions are not blocked.

## Enhancements

None.

## Constraints

- This feature takes effect only when the maximum availability mode is enabled.
- Enabling this feature may affect the RPO. If the primary node is faulty within the configured timeout window, its transactions are committed locally but not synchronized to the faulty synchronous standby nodes.
- This feature does not apply to cascaded standby nodes.

## Dependencies

This feature depends on the maximum availability mode.

# 2.12 Parallel Logical Decoding

## Availability

This feature is available since openGauss 3.0.0.

## Introduction

Multi-thread parallel decoding is supported.

## Benefits

The logical decoding performance is greatly improved and the decoding speed can be improved from 3 to 5 Mbit/s to 100 Mbit/s in standard scenarios (16-core CPU, 128 GB memory, network bandwidth > 200 Mbit/s, 10 to 100 columns in a table, 0.1 KB to 1 KB data volume in a single row, insert as the main DML operation, no flush transaction is involved, that is, number of statements in a single transaction < 4096).

## Description

When JDBC or pg\_recvlogical is used for decoding, you can set **parallel-decode-num** to a value greater than 1 and less than or equal to 20 to enable parallel decoding. In this way, one read thread, multiple decoding threads, and one sending thread are used to perform logical decoding, significantly improving the decoding speed.

## Enhancements

None.

## Constraints

1. The hardware and network environment must be normal. The size of a logical log is twice that of an Xlog. To ensure that the speed of the Xlog reaches 100 Mbit/s, the I/O bandwidth must be at least 200 Mbit/s. Resources need to be reserved for the reader, decoder, and sender threads. The number of CPU cores to be reserved must be the number of concurrent threads plus 2. For example, if there are four concurrent threads, six CPU cores need to be reserved. In actual scenarios, decoding on the standby node can meet the requirements, and no special resource reservation planning is required. To ensure that the decoding performance meets the requirements and minimize the impact on services, you are advised to set up only one parallel decoding connection on a standby node to ensure that the CPU, memory, and bandwidth resources are sufficient.
2. The value of GUC parameter **wal\_level** of the log level is **logical**.
3. The GUC parameter **max\_replication\_slots** is set to a value greater than or equal to the number of physical streaming replication slots, backup slots, and logical replication slots required by each DN.
4. The value of the decoding configuration option **parallel-decode-num** is greater than 1 and less than or equal to 20, specifying the number of concurrent decoding threads.
5. Decoding into DDL statements is not supported.
6. Decoding is not supported for column-store data and data page replication.
7. Distributed transactions cannot be decoded. Currently, decoding is performed by DN, which cannot ensure consistency in distributed transaction decoding.
8. The size of a single tuple cannot exceed 1 GB. Decoding results may be larger than inserted data. Therefore, it is recommended that the size of a single tuple be less than or equal to 500 MB.
9. Decoding compressed tables into DML statements is not supported.
10. GaussDB Kernel supports the following data types for decoding: **INTEGER**, **BIGINT**, **SMALLINT**, **TINYINT**, **SERIAL**, **SMALLSERIAL**, **BIGSERIAL**, **FLOAT**, **DOUBLE PRECISION**, **DATE**, **TIME[WITHOUT TIME ZONE]**, **TIMESTAMP[WITHOUT TIME ZONE]**, **CHAR(*n*)**, **VARCHAR(*n*)**, and **TEXT**.
11. If the SSL connection is required, the GUC parameter **ssl** must be set to **on**.
12. Interval partitioned tables cannot be replicated.
13. After a DDL statement is executed in a transaction, the DDL statement and subsequent statements are not decoded.

14. To perform decoding on the standby node, set the GUC parameter **enable\_slot\_log** to **on** on the corresponding host.
15. Currently, ultra-large CLOB decoding is not supported.
16. The same replication slot for decoding cannot be used between the primary node and standby node or between different standby nodes at the same time. Otherwise, data inconsistency will occur.
17. Do not perform operations on the replication slot on other nodes when the logical replication slot is in use. To delete a replication slot, stop decoding in the replication slot first.

## Dependencies

Decoding is performed on the standby node.

## 2.13 DCF

### Availability

This feature is available since openGauss 2.0.0.

### Introduction

Distributed consensus framework (DCF) implements data synchronization based on the Paxos algorithm. After the DCF mode is enabled, DNs support Paxos-based replication and quorum capabilities.

### Benefits

Primary DN selection and log replication are performed based on Paxos. Compression and flow control are supported during the replication to prevent high bandwidth usage. DCF provides capabilities such as log replication and cluster HA. DCF supports automatic leader election, forcible minority startup, and dynamic traffic adjustment for log replication. In addition, multiple Paxos-based roles are provided and can be adjusted. You can query the instance status in the current database.

DCF is a high-performance, highly mature, reliable, scalable, and easy-to-use independent basic library. Other systems can easily interconnect with DCF through interfaces to obtain the strong consistency, high availability, and automatic disaster recovery capabilities provided by the Paxos algorithm.

### Description

- During log replication, DCF supports log compression before transmission to reduce network bandwidth usage.
- DCF supports SSL, including TLS 1.2 and TLS 1.3. When SSL is enabled, the DN configures DCF as TLS 1.2 by default.
- DCF supports five TLS 1.3 cipher suites: TLS13-AES-256-GCM-SHA384, TLS13-CHACHA20-POLY1305-SHA256, TLS13-AES-128-GCM-SHA256, TLS13-AES-128-CCM-8-SHA256 and TLS13-AES-128-CCM-SHA256.

- DCF supports passive nodes, which do not participate in leader election and only synchronize and replay logs. When this type of nodes is heavily loaded during log synchronization, flow control is performed.
- DCF supports logger nodes, which participate in leader election and voting. However, the logger nodes only replicate DCF logs. Xlogs are not replicated and redo is not performed.
- DCF supports online switchover between follower and passive nodes. That is, without interrupting services, a node can switch over between the follower and passive roles.
- DCF supports forcible startup of the minority of DN. In case that the majority of database instances are faulty, a minority of standby DN are selected and forcibly promoted to primary, and other normal standby DN replicate logs from the primary DN.
- DCF supports automatic leader election. When the original primary DN is faulty, a new primary DN is automatically selected from the remaining standby DN on the premise that data consistency is ensured.

## Enhancements

None.

## Constraints

To use this function, you need to enable DCF during installation and deployment. In DCF mode, the majority election is used. During the installation, if the number of faulty nodes and build nodes reaches the majority, the cluster installation will fail. For example, when one primary node and two standby nodes are installed, one node fails to be installed due to insufficient memory, but the other two nodes can be started normally. A standby node will be built again. In this case, the number of build nodes and faulty nodes is 2, which is the majority, and the cluster installation will fail. During the installation, check whether the memory and disk resources are sufficient.

## Dependencies

None.

# 2.14 CM

## Availability

This feature is available since openGauss 3.0.0.

## Introduction

Cluster manager (CM) is a database management software, which consists of `cm_server` and `cm_agent`.

- `cm_agent` is a database management component deployed on each database host. It is used to start, stop, and monitor database instance processes.

- `cm_server` is a component used for managing database instances and arbitrating instances.

## Benefits

It manages and monitors the running status of functional units and physical resources in a database system, ensuring stable running of the system.

## Description

It supports customized resource monitoring and provides capabilities such as monitoring of the primary/standby database status, network communication faults, file system faults, and automatic primary/standby switchover upon faults. It also provides various database management capabilities, such as starting and stopping nodes and instances, querying database instance status, performing primary/standby switchover, and managing logs.

## Enhancements

None.

## Constraints

In scenarios where there are one primary node and one standby node, CM supports only basic capabilities, such as installation, startup, stop, and detection.

## Dependencies

None.

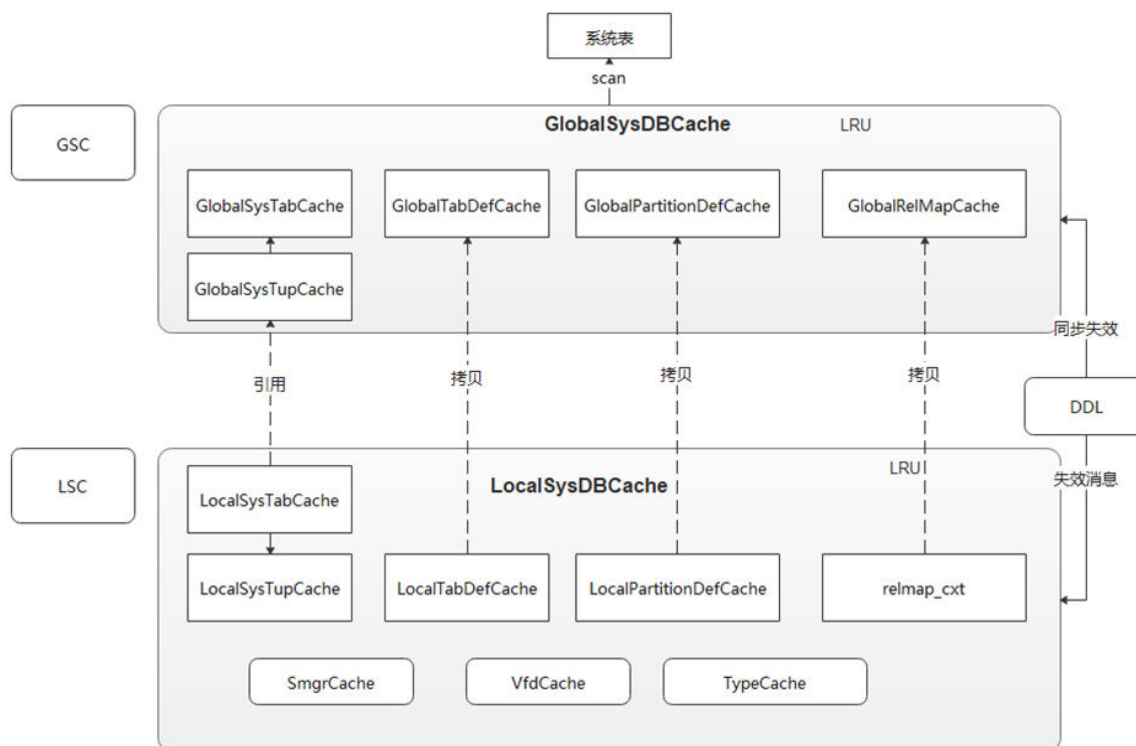
# 2.15 Global SysCache

## Availability

This feature is available since openGauss 3.0.0.

## Introduction

Global SysCache is the global and local caches of system catalog data. [Figure 2-1](#) illustrates the principle.

**Figure 2-1** Global SysCache principle

## Benefits

Global SysCache reduces the cache memory usage of database processes and improves the concurrent expansion capability of a database.

## Description

Global SysCache decouples the system cache from sessions and binds them to threads to reduce the memory usage together with the thread pool feature. In addition, it is used to improve the cache hit rate and ensure stable performance.

## Enhancements

Supports more concurrent queries.

## Constraints

- Set **enable\_global\_syscache** to **on**. You are advised to set **enable\_thread\_pool** to **on**.
- When the number of databases is large and the value of **global\_syscache\_threshold** is small, memory control cannot work properly and the performance deteriorates.
- Distributed time series tasks are not supported. The memory control and performance of these tasks are not affected by the GSC feature.
- If **wal\_level** is set to **minimal** or **archive**, the query performance of the standby server deteriorates and short connections are used.

## Dependencies

The memory reduction capability of this feature depends on the thread pool feature.

## 2.16 Using a Standby Node to Build a Standby Node

### Availability

This feature is available since openGauss 3.0.0.

### Introduction

A standby node can be built by another standby node to accelerate standby node recovery from faults. The I/O and bandwidth pressure of the primary node can be reduced.

### Benefits

When the service load is heavy, building a standby node by using the primary node increases the resource burden on the primary node. As a result, the performance of the primary node deteriorates and the build becomes slow. Building a standby node by using a standby node does not affect services on the primary node.

### Description

You can run the **gs\_ctl** command to specify a standby node to build another standby node to be repaired. For details, see "Tools Used in the Internal System > gs\_ctl" in *Tool Reference*.

### Enhancements

None.

### Constraints

A standby node can only be used to build another standby node. You can only use a specified IP address and port number to build the standby node. Before building the standby node, ensure that the logs of the standby node to be repaired are later than those of the standby node that sends data.

### Dependencies

None.



# 3 Maintainability

---

[3.1 Gray Upgrade](#)

[3.2 WDR](#)

[3.3 Slow SQL Diagnosis](#)

[3.4 Session Performance Diagnosis](#)

[3.5 System KPI-aided Diagnosis](#)

## 3.1 Gray Upgrade

### Availability

This feature is available since openGauss 2.0.0.

### Introduction

Gray upgrade supports full-service operations. All nodes can be upgraded at a time.

### Benefits

Gray upgrade provides an online upgrade mode to ensure that all nodes are upgraded without interrupting services.

### Description

Gray upgrade is an online upgrade mode that upgrades all nodes. If only the binary files of the database need to be replaced during the gray upgrade, to minimize the impact on services, the two sets of binary files exist on the same node at the same time, and the soft connection switchover mode is used to switch the process version (one intermittent disconnection within 10 seconds).

### Enhancements

None.

## Constraints

For details about the gray upgrade constraints, see "Before You Start > Upgrade Impact and Constraints" in *Upgrade Guide*.

## Dependencies

None.

## 3.2 WDR

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

The workload diagnosis report (WDR) provides database performance diagnosis reports based on the baseline performance and incremental data that reflects performance changes.

### Benefits

- The WDR is the main method for diagnosing long-term performance problems. Based on the performance baseline of a snapshot, performance analysis is performed from multiple dimensions, helping DBAs understand the system load, performance of each component, and performance bottlenecks.
- Snapshots are also an important data source for self-diagnosis and self-optimization suggestions on subsequent performance problems.

### Description

The WDR generates a performance report between two different time points based on the system performance snapshot data at these time points. The report is used to diagnose database kernel performance faults.

You can use `generate_wdr_report(...)` to generate a performance report based on two performance snapshots.

The WDR depends on the following two components:

- **Snapshot:** The performance snapshot can be configured to collect a certain amount of performance data from the kernel at a specified interval and store the data in the user tablespace. Any snapshot can be used as a performance baseline for comparison with other snapshots.
- **WDR Reporter:** This tool analyzes the overall system performance based on two snapshots, calculates the changes of more specific performance indicators between the two time points, and generates summarized and detailed performance data. For details, see [Table 3-1](#) and [Table 3-2](#).

**Table 3-1** Summarized diagnosis report

Diagnosis Type	Description
Database Stat	<p>Evaluates the load and I/O status of the current database. Load and I/O are the most important indicators of the TP system.</p> <p>The statistics include the number of sessions connected to the database, number of committed and rolled back transactions, number of read disk blocks, number of disk blocks found in the cache, number of rows returned, captured, inserted, updated, and deleted through database query, number of conflicts and deadlocks, usage of temporary files, and I/O read/write time.</p>
Load Profile	<p>Evaluates the current system load from the time, I/O, transaction, and SQL dimensions.</p> <p>The statistics include the job running elapse time, CPU time, daily transaction quality, logical and physical read volume, read and write I/O times and size, login and logout times, SQL, transaction execution volume, and SQL P80 and P95 response time.</p>
Instance Efficiency Percentages	<p>Evaluates the cache efficiency of the current system.</p> <p>The statistics include the database cache hit ratio.</p>
Events	<p>Evaluates the performance of key system kernel resources and key events.</p> <p>The statistics include the number of times that the key events of the database kernel occur and the waiting time.</p>
Wait Classes	<p>Evaluates the performance of key events in the system.</p> <p>The statistics include the release of the data kernel in the main types of wait events, such as <b>STATUS</b>, <b>LWLOCK_EVENT</b>, <b>LOCK_EVENT</b>, and <b>IO_EVENT</b>.</p>
CPU	<p>Includes time release of the CPU in user mode, kernel mode, I/O wait mode, or idle mode.</p>
IO Profile	<p>Includes the number of database I/O times, database I/O data volume, number of redo I/O times, and redo I/O volume.</p>
Memory Statistics	<p>Includes maximum process memory, used process memory, maximum shared memory, and used shared memory.</p>

**Table 3-2** Detailed diagnosis report

Diagnosis Type	Description
Time Model	Evaluates the performance of the current system in the time dimension.  The statistics include time consumed by the system in each phase, including the kernel time, CPU time, execution time, parsing time, compilation time, query rewriting time, plan generation time, network time, and I/O time.
SQL Statistics	Diagnoses SQL statement performance problems.  The statistics include normalized SQL performance indicators in multiple dimensions: elapsed time, CPU time, rows returned, tuple reads, executions, physical reads, and logical reads. The indicators can be classified into execution time, number of execution times, row activity, and cache I/O.
Wait Events	Diagnoses performance of key system resources and key time in detail.  The statistics include the performance of all key events in a period of time, including the number of events and the time consumed.
Cache IO Stats	Diagnoses the performance of user tables and indexes.  The statistics include read and write operations on all user tables and indexes, and the cache hit ratio.
Utility status	Diagnoses the background task performance.  The statistics include the performance of background tasks such as replication.
Object stats	Diagnoses the performance of database objects.  The statistics include user tables, tables on indexes, index scan activities, as well as insert, update, and delete activities, number of valid rows, and table maintenance status.
Configuration settings	Determines whether the configuration is changed.  It is a snapshot that contains all current configuration parameters.
SQL detail	Displays information about unique query text.

## Enhancements

None.

## Constraints

- The WDR snapshot collects performance data of different databases. If there are a large number of databases or tables in the database instance, it takes a long time to create a WDR snapshot.
- If WDR snapshot is performed when a large number of DDL statements are executed, WDR snapshot may fail.
- When the database is dropped, WDR snapshot may fail.

## Dependencies

None.

# 3.3 Slow SQL Diagnosis

## Availability

This feature is available since openGauss 1.1.0. The following slow SQL views have been discarded before reconstruction: `dbperf.gs_slow_query_info`, `dbperf.gs_slow_query_history`, `dbperf.global_slow_query_histroy`, and `dbperf.global_slow_query_info`.

## Introduction

Slow SQL diagnosis provides necessary information for diagnosing slow SQL statements, helping developers backtrack SQL statements whose execution time exceeds the threshold and diagnose SQL performance bottlenecks.

## Benefits

Slow SQL provides detailed information required for slow SQL diagnosis. You can diagnose performance problems of specific slow SQL statements offline without reproducing the problem. The table-based and function-based APIs help users collect statistics on slow SQL indicators and connect to third-party platforms.

## Description

Slow SQL diagnosis records information about all jobs whose execution time exceeds the threshold **`log_min_duration_statement`**.

Slow SQL provides table-based and function-based query APIs. You can query the execution plan, start time, end time, query statement, row activity, kernel time, CPU time, execution time, parsing time, compilation time, query rewriting time, plan generation time, network time, I/O time, network overhead, and lock overhead. All information is anonymized.

## Enhancements

Optimized slow SQL indicators, security (anonymization), execution plans, and query interfaces.

Run the following command to check the execution information about the SQL statements in the database instance:

```
gsql> select * from db_perf.get_global_full_sql_by_timestamp(start_timestamp, end_timestamp);
For example:
openGauss=# select * from DBE_PERF.get_global_full_sql_by_timestamp('2020-12-01 09:25:22', '2020-12-31
23:54:41');
-[ RECORD 1 ]-----
+-----+
node_name      | dn_6001_6002_6003
db_name        | postgres
schema_name    | "$user",public
origin_node    | 1938253334
user_name      | user_dj
application_name | gsql
client_addr    |
client_port    | -1
unique_query_id | 3671179229
debug_query_id  | 72339069014839210
query          | select name, setting from pg_settings where name in (?)
start_time     | 2020-12-19 16:19:51.216818+08
finish_time    | 2020-12-19 16:19:51.224513+08
slow_sql_threshold | 1800000000
transaction_id  | 0
thread_id      | 139884662093568
session_id     | 139884662093568
n_soft_parse   | 0
n_hard_parse   | 1
query_plan     | Datatype Name: dn_6001_6002_6003
               | Function Scan on pg_show_all_settings a (cost=0.00..12.50 rows=5 width=64)
               | Filter: (name = '****':text)
...

Run the following command to check the execution information about the slow SQL statements in the
database instance:
gsql> select * from db_perf.get_global_slow_sql_by_timestamp(start_timestamp, end_timestamp);
openGauss=# select * from DBE_PERF.get_global_slow_sql_by_timestamp('2020-12-01 09:25:22',
'2020-12-31 23:54:41');
-[ RECORD 1 ]-----+-----+
node_name      | dn_6001_6002_6003
db_name        | postgres
schema_name    | "$user",public
origin_node    | 1938253334
user_name      | user_dj
application_name | gsql
client_addr    |
client_port    | -1
unique_query_id | 2165004317
debug_query_id  | 72339069014839319
query          | select * from DBE_PERF.get_global_slow_sql_by_timestamp(?, ?);
start_time     | 2020-12-19 16:23:20.738491+08
finish_time    | 2020-12-19 16:23:20.773714+08
slow_sql_threshold | 10000
transaction_id  | 0
thread_id      | 139884662093568
session_id     | 139884662093568
n_soft_parse   | 10
n_hard_parse   | 8
query_plan     | Datatype Name: dn_6001_6002_6003
               | Result (cost=1.01..1.02 rows=1 width=0)
               | InitPlan 1 (returns $0)
               | -> Seq Scan on pgxc_node (cost=0.00..1.01 rows=1 width=64)
               | Filter: (nodeis_active AND ((node_type = '****':"char") OR (node_type = '****':"char"))))
...

Check the execution information about the SQL statement on the current node.
gsql> select * from statement_history;
For example:
openGauss=# select * from statement_history;
-[ RECORD 1 ]-----
+-----+
```

```
-----
db_name          | postgres
schema_name      | "$user",public
origin_node      | 1938253334
user_name        | user_dj
application_name  | gsql
client_addr      |
client_port      | -1
unique_query_id  | 3671179229
debug_query_id   | 72339069014839210
query            | select name, setting from pg_settings where name in (?)
start_time       | 2020-12-19 16:19:51.216818+08
finish_time      | 2020-12-19 16:19:51.224513+08
slow_sql_threshold | 1800000000
transaction_id   | 0
thread_id        | 139884662093568
session_id       | 139884662093568
n_soft_parse     | 0
n_hard_parse     | 1
query_plan       | Datanode Name: dn_6001_6002_6003
                  | Function Scan on pg_show_all_settings a (cost=0.00..12.50 rows=5 width=64)
                  | Filter: (name = '****':text)
```

## Constraints

- The SQL tracing information is based on the normal execution logic. The tracing information may inaccurate if SQL statements fail to be executed.
- Restarting a node may cause data loss on the node.
- If you exit a session immediately after SQL statements are executed, the session data that is not updated to the system catalog may be lost.
- The number of SQL statements to be collected is specified by a GUC parameter. If the number of SQL statements exceeds the threshold, new SQL statement execution information will not be collected.
- The maximum number of bytes of lock event details collected by a single SQL statement is specified by a GUC parameter. If the number of bytes exceeds the threshold, new lock event details will not be collected.
- The SQL statement information is updated in asynchronous mode. Therefore, after a query statement is executed, the related view function result is slightly delayed.
- Certain indicator information (such as row activities, cache I/O, and time distribution) depends on the `dbperf.statement` view. If the number of records in the view exceeds the preset size (depending on `GUC:instr_unique_sql_count`), related indicators may not be collected.
- Functions related to the `statement_history` table and the details column in the view are in binary format. To parse the detailed information, use the **`pg_catalog.statement_detail_decode(details, 'plaintext', true)`** function.
- The `statement_history` table can be queried only in the postgres database. The data in other databases is empty.
- The content of the `statement_history` table is controlled by `track_stmt_stat_level`. The default value is **`'OFF,L0'`**. The first part of the parameter indicates the full SQL statement, and the second part indicates the slow SQL statement. Slow SQL statements are recorded in the `statement_history` table only when the SQL statement execution time exceeds the value of **`log_min_duration_statement`**.

## Dependencies

None.

# 3.4 Session Performance Diagnosis

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Session performance diagnosis targets session-level performance faults.

## Benefits

- Display the latest events that consume the most resources of user sessions.
- Check the wait events that occupy the most resource-consuming SQL statements.
- Check the wait events that occupy the most resource-consuming sessions.
- Check information about the most resource-consuming users.
- Check the waiting relationship between blocked sessions.

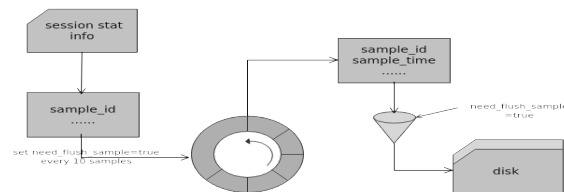
## Description

The session performance diagnosis function diagnoses performance of all active sessions in the system. As real-time collection of indicators of all active sessions has a greater impact on user load, the session snapshot technology is used to sample indicators of active sessions, and collect statistics on active sessions from the sampling. The statistics reflect the basic information, status, and resources of active sessions from the dimensions of client information, execution start time, execution end time, SQL text, wait events, and current database objects. The active session information collected based on the probability can help users diagnose which sessions consume more CPU and memory resources, which database objects are hot objects, and which SQL statements consume more key event resources in the system. In this way, users can locate faulty sessions, SQL statements, and database designs.

Session sampling data is classified into two levels, as shown in [Figure 3-1](#).

1. The first level is real-time information stored in the memory. The active session information in the latest several minutes is displayed, which has the highest precision.
2. The second level is the persistent historical information stored in disk files. It displays the historical active session information in a long period of time and is sampled from the memory data. This level is suitable for long-run statistics and analysis.



**Figure 3-1** Session performance diagnosis principle

Some application scenarios are as follows:

1. Check the blocking relationship between sessions.  
`select sessionid, block_sessionid from pg_thread_wait_status;`
2. Sample information about blocked sessions.  
`select sessionid, block_sessionid from DBE_PERF.local_active_session;`
3. Display the final blocked session.  
`select sessionid, block_sessionid, final_block_sessionid from DBE_PERF.local_active_session;`
4. Check the wait event that consumes the most resources.  
`SELECT s.type, s.event, t.count  
FROM dbe_perf.wait_events s, (  
SELECT event, COUNT(*)  
FROM dbe_perf.local_active_session  
WHERE sample_time > now() - 5 / (24 * 60)  
GROUP BY event)t WHERE s.event = t.event ORDER BY count DESC;`
5. Check the events that consume the most session resources in the last five minutes.  
`SELECT sessionid, start_time, event, count  
FROM (  
SELECT sessionid, start_time, event, COUNT(*)  
FROM dbe_perf.local_active_session  
WHERE sample_time > now() - 5 / (24 * 60)  
GROUP BY sessionid, start_time, event) as t ORDER BY SUM(t.count) OVER (PARTITION BY t.  
sessionid, start_time)DESC, t.event;`
6. Check the events that consume the most resources in the last five minutes.  
`SELECT query_id, event, count  
FROM (  
SELECT query_id, event, COUNT(*)  
FROM dbe_perf.local_active_session  
WHERE sample_time > now() - 5 / (24 * 60)  
GROUP BY query_id, event) t ORDER BY SUM(t.count) OVER (PARTITION BY t.query_id ) DESC,  
t.event DESC;`

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

## 3.5 System KPI-aided Diagnosis

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

KPIs are views of key performance indicators for kernel components or the entire system. Based on KPIs, users can learn about the real-time and historical running status of the system.

### Benefits

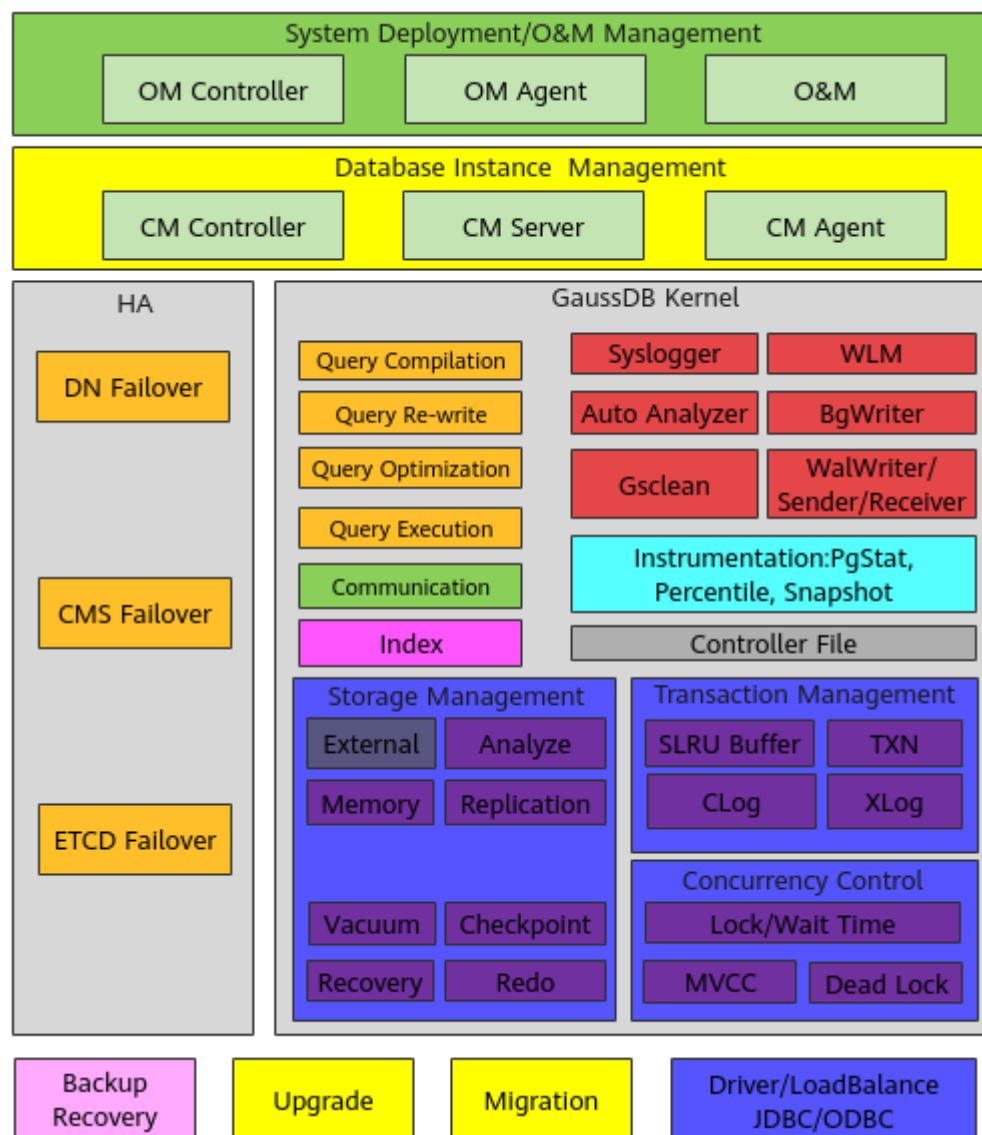
- Summarized system load diagnosis  
Precise alarms for system load exceptions (overload, stall, and SLA exceptions) and precise system load profile
- Summarized system time model diagnosis  
Instance-level and query-level time model segmentation, diagnosing the root causes of instance and query performance problems
- Query performance diagnosis  
Database-level query summary, including top SQL, SQL CPU usage, I/O consumption, execution plan, and excessive hard parsing
- Diagnosis of disk I/O, index, and buffer performance problems
- Diagnosis of connection and thread pool problems
- Diagnosis of checkpoint and redo (RTO) performance problems
- Diagnosis of system I/O, LWLock, and wait performance problems  
Diagnosis of over 60 modules and over 240 key operation performance problems
- Function-level performance monitoring and diagnosis (by GSTRACE)  
Tracing of over 50 functions at the storage and execution layers

### Description

openGauss provides KPIs of 11 categories and 26 sub-categories, covering instances, files, objects, workload, communication, sessions, threads, cache I/O, locks, wait events, and clusters.

**Figure 3-2** shows the distribution of kernel KPIs.

**Figure 3-2** Distribution of kernel KPIs



## Enhancements

None.

## Constraints

- Utility statements do not support normalization. Non-DML statements, such as CREATE, DROP, COPY, and VACUUM, are not supported.
- Currently, only the top-level normalized SQL statements are recorded. SQL statements in a stored procedure are not normalized, and only the SQL statements that call the stored procedure are recorded.

## Dependencies

None.

# 4 Database Security

---

- 4.1 Access Control Model
- 4.2 Separation of Control and Access Permissions
- 4.3 Database Encryption Authentication
- 4.4 Data Encryption and Storage
- 4.5 Database Audit
- 4.6 Network Communication Security
- 4.7 Resource Label
- 4.8 Unified Audit
- 4.9 Dynamic Data Masking
- 4.10 Row-Level Access Control
- 4.11 Password Strength Verification
- 4.12 Equality Query in a Fully-encrypted Database
- 4.13 Ledger Database Mechanism
- 4.14 Transparent Data Encryption

## 4.1 Access Control Model

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

The access control model can be used to manage users' access permissions and grant them the minimum permissions required for completing a task.

## Benefits

You can create users and grant permissions to them as needed to minimize risks.

## Description

The database provides a role-based access control model and an access control model based on the separation of duties. In the role-based access control model, database roles are classified into system administrator, monitoring administrator, O&M administrator, security policy administrator, and common user. The security administrator creates roles or user groups and grant permissions to roles. The monitoring administrator views the monitoring views or functions in **db\_perf** mode. The security policy administrator creates resource labels, anonymization policies, and unified audit policies. A user who is assigned a role has the role's permissions.

In the access control model based on the separation of duties, database roles are classified into system administrator, security administrator, audit administrator, monitoring administrator, O&M administrator, security policy administrator, and common user. The security administrator creates users, the system administrator grants permissions to users, and the audit administrator audits all user behavior.

By default, the role-based access control model is used. To switch to another mode, set the GUC parameter **enableSeparationOfDuty** to **on**.

## Enhancements

None.

## Constraints

The permissions of the system administrator are controlled by the GUC parameter **enableSeparationOfDuty**.

The database needs to be restarted when the separation of duties is enabled, disabled or switched. In addition, improper user permissions in the new model cannot be automatically identified. The database administrator needs to manually identify and rectify the fault.

## Dependencies

None.

# 4.2 Separation of Control and Access Permissions

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

The control permissions and the access permissions can be separated.

## Benefits

The control permissions of database administrators for tables need to be isolated from their access permissions to improve the data security of common users.

## Description

If multiple business departments use different database users to perform service operations and a database maintenance department at the same level uses database administrators to perform O&M operations, the business departments may require that database administrators can only perform control operations (**DROP**, **ALTER**, and **TRUNCATE**) and cannot perform access operations (**INSERT**, **DELETE**, **UPDATE**, **SELECT**, and **COPY**) without authorization. That is, the control permissions of database administrators for tables need to be isolated from their access permissions to improve the data security of common users.

In separation-of-duties mode, a database administrator does not have permissions for the tables in schemas of other users. In this case, database administrators have neither control permissions nor access permissions. This does not meet the requirements of the business departments mentioned above. Therefore, openGauss provides private users to solve the problem. That is, create private users with the **INDEPENDENT** attribute in non-separation-of-duties mode. Users with the **CREATEROLE** permission or the system administrator permission can create private users or change the attributes of common users to private users. Common users can also change their own attributes to private users.

```
openGauss=# CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "1234@abc";
```

System administrators can manage (**DROP**, **ALTER**, and **TRUNCATE**) table objects of private users but cannot access (**INSERT**, **DELETE**, **SELECT**, **UPDATE**, **COPY**, **GRANT**, **REVOKE**, and **ALTER OWNER**) the objects before being authorized.

## Enhancements

None.

## Constraints

For a table owned by a private user, grant the trigger permission of the table to other users with caution to prevent other users from using the trigger to view the data of the private user.

If permissions related to private user tables are granted to non-private users, the system administrator will obtain the same permissions.

## Dependencies

None.

# 4.3 Database Encryption Authentication

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

The client/server (C/S) mode-based client connection authentication mechanism is provided.

## Benefits

The unidirectional, irreversible hash encryption algorithm PBKDF2 is used for encryption and authentication, effectively defending against rainbow attacks.

## Description

openGauss uses a basic client connection authentication mechanism. After a client initiates a connection request, the server verifies the information and sends the information required for authentication to the client based on the verification result. The authentication information includes the salt, token, and server signature. The client responds to the request and sends the authentication information to the server. The server calls the authentication module to authenticate the client authentication information. The user password is encrypted and stored in the memory. During the entire authentication process, passwords are encrypted for storage and transmission. When the user logs in to the system next time, the hash value is calculated and compared with the key value stored on the server to verify the correctness.

## Enhancements

The message processing flow in the unified encryption and authentication process effectively prevents attackers from cracking the username or password by capturing packets.

## Constraints

None.

## Dependencies

None.

# 4.4 Data Encryption and Storage

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Imported data is encrypted before stored.

## Benefits

You can use encrypted import interfaces to encrypt sensitive information and store it in a table.

## Description

openGauss provides the encryption functions **gs\_encrypt\_aes128()** and **gs\_encrypt()**, and decryption functions **gs\_decrypt\_aes128()** and **gs\_decrypt()**. Before you import data to a certain column in a table, you can use this function to encrypt the data. The function can be called using a statement in the following format:

```
gs_encrypt_aes128(column, key), gs_encrypt (decryptstr, keystr, decrypttype)
```

In the preceding command, **key** indicates the initial password specified by the user, which is used to derive the encryption key. To encrypt an entire table, you need to write an encryption function for each column.

If a user with the required permission wants to view specific data, the user can decrypt required columns using the decryption function interface **gs\_decrypt\_aes128(column, key)**. To invoke the interface, run the following command:

```
gs_decrypt_aes128(column, key), gs_decrypt(decryptstr, keystr, decrypttype)
```

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 4.5 Database Audit

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Audit logs record user operations performed on database startup and stopping, as well as connection, DDL, DML, and DCL operations.

## Benefits

The audit log mechanism enhances the database capability of tracing unauthorized operations and collecting evidence.

## Description

Database security is essential for a database system. openGauss writes all user operations in the database into audit logs. Database security administrators can use the audit logs to reproduce a series of events that cause faults in the database



and identify unauthorized users, unauthorized operations, and the time when these operations are performed.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 4.6 Network Communication Security

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

To secure the transmission of sensitive data on the Internet, openGauss encrypts communication between the server and the client using the Secure Socket Layer (SSL) protocol.

## Benefits

The communication between your client and the server can be secured.

## Description

openGauss supports the SSL protocol. The SSL protocol is an application-layer communication protocol with high security, which is mainly used for secure web transmission. SSL contains a record layer and a transport layer. The record-layer protocol determines the encapsulation format of the transport-layer data. The transport-layer security protocol uses X.509 for authentication. The SSL protocol uses asymmetric encryption algorithms to authenticate the identities of communicating parties, and then the two parties exchange symmetric keys as communication keys. The SSL protocol effectively ensures the confidentiality and reliability of the communication between two applications and prevents the communication between a client and a server from being eavesdropped by attackers.

openGauss also supports the TLS 1.2 protocol. TLS 1.2 is a transport-layer communication protocol with high security. It consists of the TLS Record and TLS Handshake protocols. Each protocol suit has information in multiple formats. The TLS protocol is independent of application-layer protocols. Upper-layer protocols can be transparently distributed on the TLS protocol. The TLS protocol ensures the data confidentiality and integrity for both communication parties.

## Enhancements

Checking the strength of certificate signature algorithms: For low-strength signature algorithms, alarms are reported, reminding you to replace the certificate with another certificate containing a high-strength signature algorithm.

Checking the certificate validity period: If a certificate is about to expire in less than seven days, an alarm is reported, reminding you to replace the certificate on the client.

Checking certificate permissions: The certificate permissions are verified at the connection setup stage.

## Constraints

The formal certificates and keys for servers and clients shall be obtained from the Certificate Authority (CA). Assume the private key and certificate for a server are **server.key** and **server.crt**, the private key and certificate for the client are **client.key** and **client.crt**, and the CA root certificate is **cacert.pem**.

You need to enable the SSL protocol and configure the certificate and connection mode.

## Dependencies

OpenSSL

# 4.7 Resource Label

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Database resources refer to database objects, such as databases, schemas, tables, columns, views, and triggers. The more the database objects are, the more complex the classification management of database resources is. The resource label mechanism is a technology that classifies and labels database resources with certain features to implement resource classification management. After adding labels to some resources in a database, administrators can perform operations such as data audit or anonymization using the labels to implement security management on labeled database resources.

## Benefits

Proper resource labels can be used to effectively classify data objects, improve management efficiency, and simplify security policy configuration. To perform unified audit or data anonymization on a group of database resources, the administrator can allocate a resource label to these resources first. The label indicates that the database resources have a certain feature or require unified configuration of a certain policy. The administrator can directly perform operations

on the resource label, which greatly reduces the complexity of policy configuration and information redundancy as well as improves management efficiency.

## Description

The resource label mechanism selectively classifies resources in the current database. Administrators can use the following SQL syntax to create a resource label and add the label to a group of database resources:

```
CREATE RESOURCE LABEL schm_lb ADD SCHEMA(schema_for_label);  
CREATE RESOURCE LABEL tb_lb ADD TABLE(schema_for_label.table_for_label);  
CREATE RESOURCE LABEL col_lb ADD COLUMN(schema_for_label.table_for_label.column_for_label);  
CREATE RESOURCE LABEL multi_lb ADD SCHEMA(schema_for_label), TABLE(table_for_label);
```

**schema\_for\_label**, **table\_for\_label**, and **column\_for\_label** indicate the schema, table, and column to be labeled, respectively. The **schm\_lb** label is added to schema **schm\_for\_label**, **tb\_lb** is added to table **table\_for\_label**, **col\_lb** is added to column **column\_for\_label**, and **multi\_lb** is added to schema **schm\_for\_label** and table **table\_for\_label**. You can perform unified audit or dynamic data anonymization using the configured resource labels, that is, manage all labeled database resources.

Currently, resource labels support the following database resource types: schema, table, column, view, and function.

## Enhancements

None.

## Constraints

- Resource labels can be created only by a user with the **POLADMIN** and **SYSADMIN** attributes or an initial user.
- Resource labels cannot be created for temporary tables.
- Columns in the same basic table can belong to only one resource tag.

## Dependencies

None.

# 4.8 Unified Audit

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

The audit mechanism is a security management solution that can effectively deal with the attackers' repudiation. The larger the audit scope is, the more operations can be monitored and the more audit logs are generated, affecting the actual audit efficiency. The unified audit mechanism is a technology that implements efficient security audit management by customizing audit policies. After the

administrator defines the audit object and audit behaviors, if the task executed by a user is associated with an audit policy, the corresponding audit behavior is generated and the audit log is recorded. Customized audit policies can cover common user management activities, as well as DDL and DML operations, meeting routine audit requirements.

## Benefits

Audit is indispensable for routine security management. When a traditional audit mechanism is used to audit an operation, such as **SELECT**, a large number of audit logs are generated, increasing the I/O of the entire system and affecting the system performance and audit efficiency of administrators. The unified audit mechanism allows you to customize policies for generating audit logs. For example, only the operation that database account **A** queries table **a** is audited. Customized audit greatly reduces the number of generated audit logs, ensuring audit behaviors and reducing the impact on system performance. In addition, customized audit policies can improve the audit efficiency of administrators.

## Description

The unified audit mechanism customizes audit behaviors based on resource labels and classifies the supported audit behaviors into the **ACCESS** and **PRIVILEGES** classes. The SQL syntax for creating a complete audit policy is as follows:

```
CREATE RESOURCE LABEL auditlabel add table(table_for_audit1, table_for_audit2);  
CREATE AUDIT POLICY audit_select_policy ACCESS SELECT ON LABEL(auditlabel) FILTER ON ROLES(usera);  
CREATE AUDIT POLICY audit_admin_policy PRIVILEGES ALTER, DROP ON LABEL(auditlabel) FILTER ON IP(local);
```

**auditlabel** indicates the resource label in the current audit, which contains two table objects. **audit\_select\_policy** defines the audit policy for user **usera** to audit the **SELECT** operation on the objects with the **auditlabel** label, regardless of the access source. **audit\_admin\_policy** defines a local audit policy for **ALTER** and **DROP** operations on the objects with the **auditlabel** label, regardless of the user. If **ACCESS** and **PRIVILEGES** are not specified, all DDL and DML operations on objects with a resource label are audited. If no audit objects are specified, operations on all objects are audited. The addition, deletion, and modification of unified audit policies are also recorded in unified audit logs.

Currently, unified audit supports the following audit behaviors:

SQL Type	Supported operations and object types
DDL	Operations: ALL, ALTER, ANALYZE, COMMENT, CREATE, DROP, GRANT, and REVOKE SET SHOW Objects: DATABASE, SCHEMA, FUNCTION, TRIGGER, TABLE, SEQUENCE, FOREIGN_SERVER, FOREIGN_TABLE, TABLESPACE, ROLE/USER, INDEX, VIEW, and DATA_SOURCE
DML	Operations: ALL, COPY, DEALLOCATE, DELETE, EXECUTE, REINDEX, INSERT, PREPARE, SELECT, TRUNCATE, and UPDATE

## Enhancements

None.

## Constraints

- The unified audit policy must be created by a user with the **POLADMIN** or **SYSADMIN** attribute, or by the initial user. Common users do not have the permission to access the security policy system catalog and system view.
- The syntax of a unified audit policy applies to either DDL or DML operations. DDL operations and DML operations are mutually exclusive in an audit policy. A maximum of 98 unified audit policies can be configured.
- Unified audit monitors the SQL statements executed by users on the clients, but does not record the internal SQL statements of databases.
- In the same audit policy, the same resource tag can be bound to different audit behaviors, and the same behavior can be bound to different resource tags. The ALL operation type includes all operations supported by DDL or DML.
- A resource label can be associated with different unified audit policies. Unified audit outputs audit information in sequence based on the policies matched by SQL statements.
- Audit logs of unified audit policies are recorded separately. Currently, no visualized query interfaces are provided. Audit logs depend on the OS service Rsyslog and are archived through the service configuration.
- In cloud service scenarios, logs need to be stored in the OBS. In hybrid cloud scenarios, you can deploy Elasticsearch to collect CN logs and perform visualized processing.
- It is recommended that **APP** in **FILTER** be set to applications in the same trusted domain. Since a client may be forged, a security mechanism must be formed on the client when **APP** is used to reduce misuse risks. Generally, you are not advised to set **APP**. If it is set, pay attention to the risk of client spoofing.
- Taking an IPv4 address as an example, the following formats are supported:

IP Address Format	Example
Single IP address	127.0.0.1
IP address with mask	127.0.0.1 255.255.255.0
CIDR IP address	127.0.0.1/24
IP address segment	127.0.0.1-127.0.0.5

## Dependencies

None.

# 4.9 Dynamic Data Masking

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Data masking is an effective database privacy protection solution, which can prevent attackers from snooping on private data. The dynamic data masking mechanism is a technology that protects privacy data by customizing masking policies. It can effectively prevent unauthorized users from accessing sensitive information while retaining original data. After the administrator specifies the object to be anonymized and customizes a data masking policy, if the database resources queried by a user are associated with a masking policy, data is anonymized based on the user identity and masking policy to restrict attackers' access to privacy data.

## Benefits

Data privacy protection is one of the required database security capabilities. It can restrict attackers' access to privacy data, ensuring privacy data security. The dynamic data masking mechanism can protect the privacy of specified database resources by configuring masking policies. In addition, the masking policy configuration is flexible and can implement targeted privacy protection in specific user scenarios.

## Description

The dynamic data masking mechanism customizes masking policies based on resource labels. It can select masking modes based on the site requirements or customize masking policies for specific users. The SQL syntax for creating a complete masking policy is as follows:

```
CREATE RESOURCE LABEL label_for_creditcard ADD COLUMN(user1.table1.creditcard);
CREATE RESOURCE LABEL label_for_name ADD COLUMN(user1.table1.name);
CREATE MASKING POLICY msk_creditcard creditcardmasking ON LABEL(label_for_creditcard);
CREATE MASKING POLICY msk_name randommasking ON LABEL(label_for_name) FILTER ON IP(local),
ROLES(dev);
```

**label\_for\_creditcard** and **msk\_name** are the resource labels for masking, and each label is allocated to two column objects. **creditcardmasking** and **randommasking** are preset masking functions. **msk\_creditcard** specifies that the masking policy **creditcardmasking** will be applied when any user accesses resources with **label\_for\_creditcard**, regardless of the access source. **msk\_name** specifies that the masking policy **randommasking** will be applied when local user **dev** accesses resources with **label\_for\_name**. If **FILTER** is not specified, the setting takes effect for all users. Otherwise, the setting takes effect only for specified users.

The following table shows the preset masking functions:

Masking Function	Example
creditcardmasking	'4880-9898-4545-2525' will be anonymized as 'xxxx-xxxx-xxxx-2525'. This function anonymizes digits except the last four digits.
basicemailmasking	'abcd@gmail.com' will be anonymized as 'xxxx@gmail.com'. This function anonymizes text before the first @.
fullemailmasking	'abcd@gmail.com' will be anonymized as 'xxxx@xxxxx.com'. This function anonymizes text before the first dot (.) (except @).
alldigitmasking	'alex123alex' will be anonymized as 'alex000alex'. This function anonymizes only digits in the text.
shufflemasking	'hello word' will be anonymized as 'hlwoeor dl'. This weak masking function is implemented through character dislocation. You are not advised to use this function to anonymize strings with strong semantics.
randommasking	'hello word' will be anonymized as 'ad5f5ghdf5'. This function randomly anonymizes text by character.
maskall	'4880-9898-4545-2525' will be anonymized as 'xxxxxxxxxxxxxxxxxxxxxx'.

The data types supported by each masking function are as follows:

Masking Function	Supported Data Types
creditcardmasking	BPCHAR, VARCHAR, NVARCHAR, TEXT (character data in credit card format only)
basicemailmasking	BPCHAR, VARCHAR, NVARCHAR, TEXT (character data in email format only)
fullemailmasking	BPCHAR, VARCHAR, NVARCHAR, TEXT (character data in email format only)
alldigitmasking	BPCHAR, VARCHAR, NVARCHAR, TEXT (character data containing digits only)
shufflemasking	BPCHAR, VARCHAR, NVARCHAR, TEXT (text data only)
randommasking	BPCHAR, VARCHAR, NVARCHAR, TEXT (text data only)
maskall	BOOL, RELTIME, TIME, TIMETZ, INTERVAL, TIMESTAMP, TIMESTAMPTZ, SMALLDATETIME, ABSTIME, TEXT, BPCHAR, VARCHAR, NVARCHAR2, NAME, INT8, INT4, INT2, INT1, NUMRIC, FLOAT4, FLOAT8, CASH

For unsupported data types, the **maskall** function is used for data masking by default. The data of the **BOOL** type is masked as '**0**'. The **RELTIME** type is masked as '**1970**'. The **TIME**, **TIMETZ**, and **INTERVAL** types are masked as '**00:00:00.0000+00**'. The **TIMESTAMP**, **TIMESTAMPTZ**, **SMALLDATETIME**, and **ABSTIME** types are masked as '**1970-01-01 00:00:00.0000**'. The **TEXT**, **CHAR**, **BPCHAR**, **VARCHAR**, **NVARCHAR2**, and **NAME** type are masked as '**x**'. The **INT8**, **INT4**, **INT2**, **INT1**, **NUMERIC**, **FLOAT4**, **FLOAT8** types are masked as '**0**'. If the data type is not supported by **maskall**, the masking policy cannot be created. If implicit conversion is involved in the masking column, the data type after implicit conversion is used for masking. In addition, if the masking policy is applied to a data column and takes effect, operations on the data in the column are performed based on the masking result.

Dynamic data masking applies to scenarios closely related to actual services. It provides users with proper masking query APIs and error handling logic based on service requirements to prevent raw data from being obtained through credential stuffing.

## Enhancements

None.

## Constraints

- The dynamic data masking policy must be created by a user with the **POLADMIN** or **SYSDADMIN** attribute, or by the initial user. Common users do not have the permission to access the security policy system catalog and system view.
- Dynamic data masking takes effect only on data tables for which masking policies are configured. Audit logs are not within the effective scope of the masking policies.
- In a masking policy, only one masking mode can be specified for a resource label.
- Multiple masking policies cannot be used to anonymize the same resource label, except when **FILTER** is used to specify user scenarios where the policies take effect and there is no intersection between user scenarios of different masking policies that contain the same resource label. In this case, you can identify the policy that a resource label is anonymized by based on the user scenario.
- It is recommended that **APP** in **FILTER** be set to applications in the same trusted domain. Since a client may be forged, a security mechanism must be formed on the client when **APP** is used to reduce misuse risks. Generally, you are not advised to set **APP**. If it is set, pay attention to the risk of client spoofing.
- For **INSERT** or **MERGE INTO** operations with the query clause, if the source table contains anonymized columns, the inserted or updated result in the preceding two operations is the anonymized value and cannot be restored.
- When the built-in security policy is enabled, the **ALTER TABLE EXCHANGE PARTITION** statement fails to be executed if the source table is in the anonymized column.



- If a dynamic data masking policy is configured for a table, grant the trigger permission of the table to other users with caution to prevent other users from using the trigger to bypass the masking policy.
- A maximum of 98 dynamic data masking policies can be created.
- Only the preceding seven preset masking policies can be used.
- Only data with the resource labels containing the **COLUMN** attribute can be anonymized.
- Only columns in base tables can be anonymized.
- Only the data queried using **SELECT** can be anonymized.
- Taking an IPv4 address as an example, the following formats are supported:

IP Address Format	Example
Single IP address	127.0.0.1
IP address with mask	127.0.0.1 255.255.255.0
CIDR IP address	127.0.0.1/24
IP address segment	127.0.0.1-127.0.0.5

## Dependencies

None.

## 4.10 Row-Level Access Control

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

The row-level access control feature enables database access control to be accurate to each row of data tables. When different users perform the same SQL query operation, the read results may be different.

### Benefits

When different users perform the same SQL query operation, the read results may be different.

### Description

You can create an RLS policy for a data table. The policy defines an expression that takes effect only for specific database users and SQL operations. When a

database user accesses the data table, if a SQL statement meets the specified row-level security policies of the data table, the expressions that meet the specified condition will be combined by using **AND** or **OR** based on the attribute type (**PERMISSIVE** | **RESTRICTIVE**) and applied to the execution plan in the query optimization phase.

Row-level access control is used to control the visibility of row-level data in tables. By predefining filters for data tables, the expressions that meet the specified condition can be applied to execution plans in the query optimization phase, which will affect the final execution result. Currently, the SQL statements that can be affected include **SELECT**, **UPDATE**, and **DELETE**.

## Enhancements

None.

## Constraints

- Row-level access control policies can be applied only to **SELECT**, **UPDATE**, and **DELETE** operations and cannot be applied to **INSERT** and **MERGE** operations.
- Row-level access control policies can be defined for row-store tables, row-store partitioned tables, column-store tables, column-store partitioned tables, replication tables, unlogged tables, and hash tables. Row-level access control policies cannot be defined for HDFS tables, foreign tables, and temporary tables.
- Row-level access control policies cannot be defined for views.
- A maximum of 100 row-level access control policies can be defined for a table.
- Initial users and system administrators are not affected by row-level access control policies.
- If a dynamic data masking policy is configured for a table that has the row-level access control policies defined, grant the trigger permission of the table to other users with caution to prevent other users from using the trigger to bypass the masking policy.

## Dependencies

None.

# 4.11 Password Strength Verification

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Verifies the password strength when users access the database.

## Benefits

Users cannot set passwords with low strength to enhance customer data security.

## Description

You need to specify a password when initializing a database, creating a user, or modifying a user. The password must meet the strength requirements. Otherwise, the system prompts you to enter the password again. Password complexity requirements:

- Minimum number of uppercase letters (A-Z) (**password\_min\_uppercase**)
- Minimum number of lowercase letters (a-z) (**password\_min\_lowercase**)
- Minimum number of digits (0-9) (**password\_min\_digital**)
- Minimum number of special characters (**password\_min\_special**)
- Minimum password length (**password\_min\_length**)
- Maximum password length (**password\_max\_length**)
- A password must contain at least three types of the characters (uppercase letters, lowercase letters, digits, and special characters).
- A password is case insensitive and cannot be the username or the username spelled backwards.
- A new password cannot be the current password and the current password spelled backwards.
- It must be a strong password.

### NOTE

Weak passwords are weak passwords that are easy to crack. The definition of weak passwords may vary with users or user groups. Users can define their own weak passwords.

If parameter **password\_policy** is set to **1**, the default password complexity rule is used to check passwords.

Passwords in the weak password dictionary are stored in the **gs\_global\_config** system catalog (the record whose name field is **weak\_password** is the stored weak password). When a user is created or modified, the password set by the user is compared with the password stored in the weak password dictionary. If the password is matched, a message is displayed, indicating that the password is weak and the password fails to be set.

The weak password dictionary is empty by default. You can add or delete weak passwords using the following syntax:

```
CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('password1'), ('password2');  
DROP WEAK PASSWORD DICTIONARY;
```

In the preceding statement, **password1** and **password2** are weak passwords prepared by users. After the statement is executed successfully, the passwords are saved to the weak password system catalog.

When a user attempts to run the CREATE WEAK PASSWORD DICTIONARY statement to insert a weak password that already exists in the table, only one weak password is retained in the table.

The DROP WEAK PASSWORD DICTIONARY statement clears weak passwords in the entire system catalog.

The gs\_global\_config system catalog does not have a unique index. You are not advised to use the COPY FROM statement to copy the same data to the gs\_global\_config system catalog.

To audit weak password operations, set the third bit of the value of the **audit\_system\_object** parameter to **1**.

## Enhancements

In openGauss 1.1.0, the weak password dictionary function is implemented.

## Constraints

- Initial users, system administrators, and security administrators can view, add, and delete weak password dictionaries.
- Common users can view but cannot add or delete weak password dictionaries.

## Dependencies

None.

# 4.12 Equality Query in a Fully-encrypted Database

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

The encrypted database aims to protect privacy throughout the data lifecycle. In this way, data is always in ciphertext during transmission, computing, and storage regardless of the service scenario and environment. After the data owner encrypts data on the client and sends the encrypted data to the server, no attacker can obtain valuable information even if the attacker steals user data by exploiting system vulnerabilities. In this way, data privacy is protected.

## Benefits

The entire service data flow is in ciphertext during data processing, so the following can be implemented by using a fully-encrypted database:

1. Protect data privacy and security throughout the lifecycle on the cloud. Attackers cannot obtain valid information from the database server regardless of the data status.
2. Help cloud service providers obtain third-party trust. Users, including service administrators and O&M administrators in enterprise service scenarios and application developers in consumer cloud services, can keep keys by themselves so that users with high permissions cannot obtain valid data.
3. Enable cloud databases to better comply with personal privacy protection laws and regulations with the help of the fully-encrypted database.

## Description

From the perspective of users, the encrypted equality query functions are divided into three parts, which are implemented by the newly added KeyTool and the enhanced openGauss gsql client.

First, this feature provides the client key management function. Users can use KeyTool to generate, destroy, and update CMKs, and import and export keys. With the import and export functions of KeyTool, CMKs can be transmitted between different clients. In addition, the KeyTool implements key management on a single client. By configuring management files, you can store and update keys.

In addition, this feature provides the key creation and encrypted table creation functions. The SQL syntax CREATE CLINET MASTER KEY and CREATE COLUMN ENCRYPTION KEY are added to record and manage CMK and CEK metadata in the database. The CMK and CEK information is recorded in the new system catalog. The CREATE TABLE syntax is extended to specify a column encryption key and encryption algorithm for each sensitive information column in a table, facilitating subsequent ciphertext data storage.

This feature supports the encrypted equality query function, which is the core of the entire feature. Although users are unaware of the ciphertext query, the query of sensitive data is restricted by the specifications of the current encrypted equality query.

From the overall perspective, this feature is used to store and manage data based on sensitive data protection requirements and implement query tasks based on ciphertext data.

## Enhancements

None.

## Constraints

- Data is encrypted at the column level, and encryption policies cannot be differentiated by row level.
- Except the RENAME operation, the ALTER TABLE syntax cannot be used to change columns in an encrypted table (including the conversion between encrypted and unencrypted columns). The ADD and DROP operations can be used to add and delete encrypted columns, respectively.
- The CHECK(COLUMN IS NOT NULL) syntax can be used, but most check constraint syntax cannot be set for encrypted columns.
- When **support\_extended\_features** is set to **off**, primary key and unique cannot be used for encrypted columns. When **support\_extended\_features** is set to **on**, only primary key and unique can be used for encrypted columns.
- Different data types cannot be implicitly converted.
- The set operation cannot be performed between ciphertexts of different data types.
- The encrypted column cannot be of the array type.
- Range partitioning cannot be created for encrypted columns.
- Only the repeat and empty\_blob() functions can be used to encrypt columns.

- The current version supports only gsql and JDBC (deployed on a Linux OS) clients. Other clients such as ODBC do not support encrypted equality query.
- When a JDBC client is used, this feature cannot be enabled together with the load balancing or automatic primary node selection feature.
- Data can only be imported to the encrypted table by running **copy from stdin, \copy, or insert into values (...)** on the client.
- Copying an encrypted table to a file is not supported.
- The system does not support encrypted queries, such as sorting, range query, and fuzzy query, except equality query.
- The encrypted syntax of stored procedures for some functions is supported. For details about the constraints, see "Encrypted Functions and Stored Procedures" in the *Developer Guide*.
- Non-encrypted table data cannot be inserted into encrypted table data using the **INSERT INTO... SELECT... or MERGE INTO** syntax.
- For a request in connection state, the CEK information change on the server can be detected only after the cache update operation is triggered (for example, the user is changed or the encrypted column fails to be decrypted) and the connection is re-established.
- Encrypted equality query is not supported on columns encrypted using the random encryption algorithm.
- An error is reported if the two attribute conditions used for comparison in the encrypted equality query use different data encryption keys.
- Encrypted equality query is not supported in time series tables and foreign tables. The ustore storage engine is not supported.
- If the database service configuration (such as the pg\_settings system catalog, permission, key, and encrypted column) is changed, you need to re-establish a JDBC connection to make the configuration take effect.
- Multiple SQL statements cannot be executed at the same time. This constraint does not apply to the scenario where the INSERT INTO statement is executed in multiple batches.
- Encrypted equality query supports the following data types:

Category	Type	Description
Integer types	tinyint/tinyint(n)	Tiny integer, which is the same as int1.
	smallint	Small integer, which is the same as int2.
	int4	Common integer.
	binary_integer	Oracle compatibility type. Generally, the value is an integer.
	bigint/bigint(n)	Big integer, which is the same as int8.
Numeric data types	numeric(p,s)	A number with the precision <b>p</b> .
	number	Oracle compatibility type, which is the same as numeric(p,s).

Floating point types	float4	Single-precision floating point.
	float8	Double-precision floating point.
	double precision	Double-precision floating point.
Character data types	char/char(n)	Fixed-length character string. If the length is insufficient, add spaces. The default precision is 1.
	varchar(n)	Variable-length character string, where <b>n</b> indicates the maximum number of bytes.
	text	Text type.
	varchar2(n)	Oracle compatibility type, which is the same as varchar(n).
	clob	Character large object.
Binary data types	bytea	Variable-length binary string.
	blob	Binary large object.

## Dependencies

None.

## 4.13 Ledger Database Mechanism

### Availability

This feature is available since openGauss 2.1.0.

### Introduction

The ledger database feature includes adding the verification information to a tamper-proof table specified by a user and recording the user's data operation history. The consistency between the data and operation history is checked to ensure that the user data cannot be maliciously tampered with. When a user performs DML operations on a tamper-proof table, the system adds a small amount of additional row-level verification information to the table and records the SQL statements and data change history. The feature provides a verification API for users to check whether the data in the tamper-proof table is consistent with the operation information recorded by the system.

### Benefits

The ledger database provides user data operation records, historical data change records, and easy-to-use consistency verification API to help users check whether sensitive information in the database is maliciously tampered with at any time, effectively improving the tamper-proof capability of the database.

## Description

The ledger database uses the ledger schema to isolate common tables from tamper-proof user tables. If a row-store table created in the ledger schema has the tamper-proof attribute, it is a tamper-proof user table. When data is inserted into a tamper-proof user table, the system automatically generates a small amount of row-level verification information. When a user executes DML, the system records user operations in the global blockchain table (GS\_GLOBAL\_CHAIN) and records data changes in the historical table corresponding to the user table. The data in operation records, data change records, and the user table must be the same. The ledger database provides a high-performance verification API for users to verify data consistency. If the consistency verification fails, the data may be tampered with. In this case, contact the audit administrator to trace the operation history.

## Enhancements

None.

## Constraints

- In tamper-proof schema, row-store tables are tamper-proofing, whereas temporary tables, unlogged tables, column-store tables, and time series tables are not.
- The structure of the tamper-proof user table cannot be modified. The tamper-proof tables cannot be truncated. The tamper-proof user table cannot be switched to a common schema. The non-tamper-proof table cannot be switched to the tamper-proof schema.
- If the tamper-proof table is a partitioned table, operations such as exchange partition, drop partition and truncate partition are not supported.
- Functions and triggers cannot be used to modify data in a tamper-proof user table.
- When a tamper-proof user table is created, the column named **hash** cannot exist.
- Common users can call the tampering verification API to verify only tables that they have the permission to query.
- Only the audit administrator and initial user can query the global blockchain table and tables in BLOCKCHAIN schema. Common users do not have the permission to access and all users do not have the permission to modify the tables.
- According to the naming rules of historical tables, if the name of the schema or table to be created ends or starts with an underscore (\_), the name of the corresponding historical table may conflict with that of an existing table. In this case, you need to rename the table.

## Dependencies

None.



## 4.14 Transparent Data Encryption

### Availability

This feature is available since openGauss 2.1.0.

### Introduction

Transparent data encryption (TDE) encrypts data when the database writes the data to the storage medium and automatically decrypts the data when reading the data from the storage medium. This prevents attackers from reading data in the data file without database authentication, solving the static data leakage problem. This function is almost transparent to the application layer. You can determine whether to enable the transparent data encryption function as required.

### Benefits

To prevent attackers from reading data files without authentication, you can use the transparent data encryption function to encrypt data files in the database. This ensures that users can read decrypted data only after starting and connecting to the database properly.

### Description

The three-layer key structure is used to implement the key management mechanism, including the root key (RK), cluster master key (CMK), and data encryption key (DEK). CMKs are encrypted and protected by RKs, and DEKs are encrypted and protected by CMKs. DEKs are used to encrypt and decrypt user data. Each table corresponds to a DEK.

Table-level encryption is supported. When creating a table, you can specify whether to encrypt the table and the encryption algorithm to be used. The encryption algorithm can be AES\_128\_CTR or SM4\_CTR, which cannot be changed once specified. If an encrypted table is created, the database automatically applies for a DEK for the table and saves the encryption algorithm, key ciphertext, and corresponding CMK ID in the **reloptions** column of the pg\_class system catalog in keyword=value format.

You can switch an encrypted table to a non-encrypted table or switch a non-encrypted table to an encrypted table. If the encryption function is not enabled when a table is created, the table cannot be switched to an encrypted table.

For encrypted tables, DEK rotation is supported. After the key rotation, the data encrypted using the old key is decrypted using the old key, and the newly written data is encrypted using the new key. The encryption algorithm is not changed during key rotation.

### Enhancements

None.

## Constraints

The current version interconnects with HUAWEI CLOUD KMS to support table-level key storage and row-store table encryption. The specifications are as follows:

- Encryption of a row-store table stored as a heap is supported.
- Column-store encryption, materialized view encryption, and ustore storage engine encryption are not supported.
- Indexes, sequences, Xlogs, MOTs, and system catalogs cannot be encrypted.
- You can specify an encryption algorithm when creating a table. Once specified, the encryption algorithm cannot be changed. If **enable\_tde** is set to **on** but the encryption algorithm **encrypt\_algo** is not specified when a table is created, the AES-128-CTR encryption algorithm is used by default.
- If the encryption function is not enabled or the encryption algorithm is not specified when a table is created, the table cannot be switched to an encrypted table.
- For a table that has been assigned an encryption key, switching between the encrypted and unencrypted states of the table does not change the key or encryption algorithm.
- Data key rotation is supported only when the table encryption function is enabled.
- Cross-region primary/standby synchronization of multiple copies in a single cluster is not supported. Cross-region scaling of a single cluster is not supported. Cross-region backup and restoration, cluster DR, and data migration are not supported.
- In hybrid cloud scenarios, if the HUAWEI CLOUD KMS and management plane functions are used, transparent data encryption is supported. Other KMS services are not supported if their APIs are incompatible.
- The query performance of encrypted tables is lower than that of non-encrypted tables. If high performance is required, exercise caution when enabling the encryption function.

## Dependencies

The key management service is provided by the external KMS. The current version can interconnect with HUAWEI CLOUD KMS.

# 5 Enterprise-Level Features

---

- 5.1 Support for Functions and Stored Procedures
- 5.2 SQL Hints
- 5.3 Full-Text Indexing
- 5.4 Copy Interface for Error Tolerance
- 5.5 Partitioning
- 5.6 Support for Advanced Analysis Functions
- 5.7 Materialized View
- 5.8 HyperLogLog
- 5.9 Creating an Index Online
- 5.10 Autonomous Transaction
- 5.11 Global Temporary Table
- 5.12 Pseudocolumn ROWNUM
- 5.13 Stored Procedure Debugging
- 5.14 JDBC Client Load Balancing and Read/Write Isolation
- 5.15 In-place Update Storage Engine
- 5.16 Publication-Subscription
- 5.17 Foreign Key Lock Enhancement
- 5.18 Data Compression in OLTP Scenarios

## 5.1 Support for Functions and Stored Procedures

### Availability

This feature is available since openGauss 1.1.0.

## Introduction

Functions and stored procedures are important database objects. They encapsulate SQL statement sets used for certain functions so that the statements can be easily invoked.

## Benefits

1. Allows customers to modularize program design and encapsulate SQL statement sets, easy to invoke.
2. Caches the compilation results of stored procedures to accelerate SQL statement set execution.
3. Allows system administrators to restrict the permission for executing a specific stored procedure and controls access to the corresponding type of data. This prevents access from unauthorized users and ensures data security.

## Description

openGauss supports functions and stored procedures compliant with the SQL standard. The stored procedures are compatible with certain mainstream stored procedure syntax, improving their usability.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 5.2 SQL Hints

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

SQL hints can be used to override execution plans.

## Benefits

Improves SQL query performance.

## Description

In plan hints, you can specify a join order; join, stream, and scan operations, the number of rows in a result, and redistribution skew information to tune an execution plan, improving query performance.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 5.3 Full-Text Indexing

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

openGauss full-text indexing allows documents to be preprocessed and facilitates subsequent search.

## Benefits

openGauss full-text indexing provides the capability to identify natural-language documents that satisfy a query and sort them by relevance.

## Description

The preprocessing process of creating a full-text index includes:

- Parsing documents into tokens  
It is useful to identify various classes of tokens, for example, numbers, words, compound words, and email addresses, so that they can be processed differently. In principle, token classes depend on the specific application, but for most purposes it is adequate to use a predefined set of classes.
- Converting tokens into lexemes  
A lexeme is a string, just like a token, but it has been normalized so that different forms of the same word are made alike. For example, normalization almost always includes folding upper-case letters to lower-case, and often involves removal of suffixes (such as **s** or **es** in English). This allows searches to find variant forms of the same word, without entering all the possible variants. Also, this step typically eliminates stop words, which are so common and usually useless for searching. (In short, tokens are raw fragments of the

document text, while lexemes are words that are believed useful for indexing and searching.) openGauss uses dictionaries to perform this step and provides various standard dictionaries.

- Storing preprocessed documents optimized for searching

For example, each document can be represented as a sorted array of normalized lexemes. Along with the lexemes, it is often desirable to store positional information for proximity ranking. Therefore, a document that contains a more "dense" area of query words is assigned with a higher rank than the one with scattered query words. Dictionaries allow fine-grained control over how tokens are normalized. With appropriate dictionaries, you can define stop words that should not be indexed.

## Enhancements

None.

## Constraints

The current limitations of openGauss's text search features are:

- The length of each lexeme must be less than 2 KB.
- The length of a **tsvector** (lexemes + positions) must be less than 1 MB.
- Position values in **tsvector** must be greater than 0 and less than or equal to 16383.
- No more than 256 positions per lexeme. Excessive positions, if any, will be discarded.

## Dependencies

None.

# 5.4 Copy Interface for Error Tolerance

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Certain errors that occur during the copy process are imported to a specified error table without interrupting the process.

## Benefits

Refine the copy function and improve the tolerance and robustness to common errors such as invalid formats.

## Description

openGauss provides the encapsulated copy error tables for creating functions and allows users to specify error tolerance options when using the **Copy From**

statement. In this way, errors related to parsing, data format, and character set during the execution of the **Copy From** statement are recorded in the error table instead of being reported and interrupted. Even if a small amount of data in the target file of **Copy From** is incorrect, the data can be imported to the database. You can locate and rectify the fault in the error table later.

## Enhancements

None

## Constraints

For details, see "Importing Data > Running the COPY FROM STDIN Statement to Import Data > Handling Import Errors" in the *Developer Guide*.

## Dependencies

None

# 5.5 Partitioning

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Data is partitioned horizontally on a node using a specified policy. This operation splits a table into multiple partitions that are not overlapped.

## Benefits

In common scenarios, a partitioned table has the following advantages over a common table:

- High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
- High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
- Balanced I/O: Partitions can be mapped to different disks to balance I/O and improve the overall system performance.

## Description

Currently, the openGauss database supports range partitioned tables, list partitioned tables, hash partitioned tables, interval partitioned tables, and level-2 partitioned tables.

- In a range partitioned table, data within a certain range is mapped to each partition. The range is determined by the partition key specified when the partitioned table is created. This partitioning mode is most commonly used.

With the range partitioning function, the database divides a record, which is to be inserted into a table, into multiple ranges using one or multiple columns and creates a partition for each range to store data. Partition ranges do not overlap.

- In a list partitioned table, data is mapped to each partition based on the key values contained in each partition. The key values contained in a partition are specified when the partition is created.

The list partitioning function divides the key values in the records to be inserted into a table into multiple lists (the lists do not overlap in different partitions) based on a column of the table, and then creates a partition for each list to store the corresponding data.

- In a hash partitioned table, data is mapped to each partition using the hash algorithm, and each partition stores records with the same hash value.

The hash partitioning function uses the internal hash algorithm to divide records to be inserted into a table into partitions based on a column of the table.

- Interval partitioning is a special type of range partitioning. Compared with range partitioning, interval partitioning defines the interval value. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.
- A level-2 partitioned table is based on level-1 partition. Its partitioning scheme is a combination of two level-1 partitioning schemes. Currently, the level-2 partitioned table supports nine partitioning policies combining range partitioning, list partitioning, and hash partitioning.

If you specify the **PARTITION** parameter when running the **CREATE TABLE** statement, data in the table will be partitioned. Users can modify partition keys as needed during table creation to make the query result stored in the same or least partitions (called partition pruning), obtaining consecutive I/O to improve the query performance.

In actual services, time is often used to filter query objects. Therefore, you can select the time column as the partition key. The key value range can be adjusted based on the total data volume and the data volume queried at a time.

## Enhancements

Range partitioned tables can be combined.

## Constraints

None.

## Dependencies

None.



## 5.6 Support for Advanced Analysis Functions

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

None.

### Benefits

Window functions are provided for advanced data analysis and processing. The window function groups the data in a table in advance. Each row belongs to a specific group. Then, a series of association analysis calculations are performed on the group. In this way, some attributes of each tuple in the set and association information with other tuples can be mined.

### Description

The following uses an example to describe the window analysis function: Compare the salary of each person in a department with the average salary of the department.

```
SELECT depname, empno, salary, avg(salary) OVER (PARTITION BY depname) FROM empsalary;
depname | empno | salary | avg
-----+-----+-----+-----
develop | 11 | 5200 | 5020.0000000000000000
develop | 7 | 4200 | 5020.0000000000000000
develop | 9 | 4500 | 5020.0000000000000000
develop | 8 | 6000 | 5020.0000000000000000
develop | 10 | 5200 | 5020.0000000000000000
personnel | 5 | 3500 | 3700.0000000000000000
personnel | 2 | 3900 | 3700.0000000000000000
sales | 3 | 4800 | 4866.6666666666666667
sales | 1 | 5000 | 4866.6666666666666667
sales | 4 | 4800 | 4866.6666666666666667
(10 rows)
```

The analysis function **avg(salary) OVER (PARTITION BY depname)** easily calculates each employee's salary and the average salary of the department.

Currently, the system supports the following analysis functions: **row\_number()**, **rank()**, **dense\_rank()**, **percent\_rank()**, **cume\_dist()**, **ntile()**, **lag()**, **lead()**, **first\_value()**, **last\_value()**, and **nth\_value()**. For details about functions and statements, see "SQL Reference > Functions and Operators > Window Functions" in the *Developer Guide*.

### Enhancements

None.

### Constraints

None.

## Dependencies

None.

# 5.7 Materialized View

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

A materialized view is a special physical table, which is relative to a common view. A common view is a virtual table and has many application limitations. Any query on a view is actually converted into a query on an SQL statement, and performance is not actually improved. The materialized view actually stores the results of the statements executed by the SQL statement, and is used to cache the results.

## Benefits

The materialized view function is used to improve query efficiency.

## Description

Full materialized views and incremental materialized views are supported. Full materialized views can only be updated in full mode. Incremental materialized views can be updated asynchronously. You can run statements to update new data to materialized views.

## Enhancements

None.

## Constraints

Only simple filter queries and UNION ALL statements are supported for base tables.

## Dependencies

None.

# 5.8 HyperLogLog

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

HyperLoglog (HLL) is used to count the number of distinct values.

## Benefits

Improves AP/TP query performance.

## Description

HLL is an approximation algorithm for efficiently counting the number of distinct values in a dataset. It features faster computing and lower space usage. You only need to store HLL data structures instead of datasets. When new data is added to a dataset, make hash calculation on the data and insert the result to an HLL. Then, you can obtain the final result based on the HLL.

HLL has advantages over others in the computing speed and storage space requirement. In terms of time complexity, the Sort algorithm needs to sort at least  $O(n \log n)$  time. Although the Hash algorithm can obtain the result by scanning the entire table  $O(n)$  time, the storage space is as follows: Both the Sort and Hash algorithms need to store the original data before collecting statistics, which consumes a large amount of storage space. For the HLL, the original data does not need to be stored, and only the HLL data structure needs to be maintained. Therefore, the occupied space is always at the 1280-byte constant level.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 5.9 Creating an Index Online

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

Uses the CREATE INDEX CONCURRENTLY syntax to create indexes online without blocking DML.

## Benefits

When creating an index, you can specify the CONCURRENTLY keyword to ensure that the DML and online services are not blocked during the index creation.

## Description

A normal CREATE INDEX acquires exclusive lock on the table on which the index depends, blocking other accesses until the index drop can be completed. If the CONCURRENTLY keyword is specified, the ShareUpdateExclusiveLock lock is added to the table so that DML is not blocked during the creation.

This keyword is specified when an index is created online. The entire table needs to be scanned twice and built. When the table is scanned for the first time, an index is created and the read and write operations are not blocked. During the second scan, changes that have occurred since the first scan are merged and updated. The table needs to be scanned and built twice, and all existing transactions that may modify the table must be completed. This means that the creation of the index takes a longer time than normal. In addition, the CPU and I/O consumption also affects other services.

## Enhancements

None

## Constraints

- Only one index name can be specified when an index is created online.
- The CREATE INDEX statement can be run within a transaction, but CREATE INDEX CONCURRENTLY cannot.
- Column-store tables, partitioned tables, and temporary tables do not support CREATE INDEX CONCURRENTLY.

## Dependencies

None

# 5.10 Autonomous Transaction

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

An autonomous transaction is a type of transaction in which the commit of a sub-transaction is not affected by the commit or rollback of the main transaction.

## Benefits

This feature meets diversified application scenarios.

## Description

In an autonomous transaction, a specified type of SQL statements are executed in an independent transaction context during the execution of the main transaction.

The commit and rollback operations of an autonomous transaction are not affected by the commit and rollback operations of the main transaction.

User-defined functions and stored procedures support autonomous transactions.

A typical application scenario is as follows: A table is used to record the operation information during the main transaction execution. When the main transaction fails to be rolled back, the operation information recorded in the table cannot be rolled back.

## Enhancements

None.

## Constraints

- A trigger function does not support autonomous transactions.
- In the autonomous transaction block of a function or stored procedure, static SQL statements do not support variable transfer.
- Autonomous transactions do not support nesting.
- A function containing an autonomous transaction does not support the return value of parameter transfer.
- A stored procedure or function that contains an autonomous transaction does not support exception handling.

## Dependencies

None.

# 5.11 Global Temporary Table

## Availability

This feature is available since openGauss 1.1.0.

## Introduction

A temporary table does not guarantee persistency. Its life cycle is usually bound to a session or transaction, which can be used to store temporary data during processing and accelerate query.

## Benefits

This feature improves the expression capability and usability of temporary tables.

## Description

The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of a session are isolated from those of another session. Each session can only view and modify the data submitted by itself.

Global temporary tables have two schemas: ON COMMIT PRESERVE ROWS and ON COMMIT DELETE ROWS. In session-based ON COMMIT PRESERVE ROWS schema, user data is automatically cleared when a session ends. In transaction-based ON COMMIT DELETE ROWS schema, user data is automatically cleared when the commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with **pg\_temp\_** when creating a global temporary table.

## Enhancements

The processing of the global temporary table is added based on the local temporary table.

## Constraints

- Parallel scanning is not supported.
- Temp tablespace is not supported.
- Partitions are not supported.
- GIST indexes are not supported.
- The user-defined statistics **pg\_statistic\_ext** is not supported.
- ON COMMIT DROP is not supported.
- Hash bucket cluster storage is not supported.
- Row store is not supported.

## Dependencies

None

# 5.12 Pseudocolumn ROWNUM

## Availability

This feature is available since openGauss 1.0.1.

## Introduction

ROWNUM is a sequence number generated for each record in the query result. The sequence number starts from 1 and is unique.

## Benefits

- This feature is compatible with Oracle features, facilitating database migration.
- Similar to the LIMIT feature, this feature can filter out the first  $n$  records in the result set.

## Description

ROWNUM (pseudocolumn), which is used to label the records that meet conditions in the SQL query in sequence. In the query result, the value of

**ROWNUM** in the first line is **1**, the value of **ROWNUM** in the second line is **2**, and so on. The value of **ROWNUM** in the  $n$ th line is  $n$ . This feature is used to filter the first  $n$  rows of data in the query result set, which is similar to the LIMIT function in openGauss.

## Enhancements

During internal execution, the optimizer rewrites ROWNUM into LIMIT to accelerate the execution speed.

## Constraints

- Do not use the pseudocolumn ROWNUM as an alias to avoid ambiguity in SQL statements.
- Do not use ROWNUM when creating an index. Bad example: **create index index\_name on table(rownum);**
- Do not use ROWNUM as the default value when creating a table. Bad example: **create table table\_name(id int default rownum);**
- Do not use ROWNUM as an alias in the WHERE clause. Bad example: **select rownum rn from table where rn < 5;**
- Do not use ROWNUM when inserting data. Bad example: **insert into table values (rownum,'blue')**
- Do not use ROWNUM in a table-less query. Bad example: **select \* from (values(rownum,1)), x(a,b);**
- If the HAVING clause contains ROWNUM (and is not in the aggregate function), the GROUP BY clause must contain ROWNUM (and is not in the aggregate function).

## Dependencies

None.

# 5.13 Stored Procedure Debugging

## Availability

This feature was introduced in openGauss 1.0.0. After the third-party library code directory structure was adjusted, this feature was temporarily deleted and is now available since openGauss 1.1.0.

## Introduction

This feature provides a group of APIs for debugging stored procedures, such as breakpoint debugging and variable printing.

## Benefits

This feature improves user experience in developing stored procedures based on openGauss.

## Description

Stored procedures are important database objects. They encapsulate SQL statement sets used for certain functions so that the statements can be easily invoked. A stored procedure usually contains many SQL statements and procedural execution structures, depending on the service scale. However, writing a large stored procedure is usually accompanied by logic bugs. It is difficult or even impossible to find the bugs by only executing the stored procedure. Therefore, a debugging tool is required.

The stored procedure debugging tool provides a group of debugging APIs to enable the stored procedure to be executed step by step. During the execution, you can set breakpoints and print variables so that SQL developers can detect and correct errors in time and develop functions more efficiently and with high quality.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 5.14 JDBC Client Load Balancing and Read/Write Isolation

## Availability

This feature is available since openGauss 2.1.0.

## Introduction

The JDBC client provides load balancing and read/write isolation capabilities.

## Benefits

Load balancing and read/write isolation can be configured on the JDBC client.

## Description

The IP addresses and port numbers of multiple nodes on the client are configured to adapt to HA switchover between multiple AZs and remote DR switchover. The connection-level read/write isolation configuration is supported. Preferentially connecting to read-only nodes is supported. Multiple read-only nodes are evenly distributed.



## Enhancements

None.

## Constraints

None.

## Dependencies

None.

# 5.15 In-place Update Storage Engine

## Availability

This feature is available since openGauss 2.1.0.

## Introduction

The in-place update storage engine is a new storage mode added to openGauss. The row storage engine used by the earlier versions of openGauss is in append update mode. The append update has good performance in addition, deletion, and HOT (Heap Only Tuple) update (that is, update on the same page) in the service. However, in a non-HOT UPDATE scenario across data pages, garbage collection is not efficient. The Ustore storage engine can solve this problem.

## Benefits

The in-place update storage engine can effectively reduce storage space occupation after tuples are updated for multiple times.

## Description

The in-place update storage engine solves the problems of space expansion and large tuples of the Append update storage engine. The design of efficient rollback segments is the basis of the in-place update storage engine.

## Enhancements

None.

## Constraints

None.

## Dependencies

None.

## 5.16 Publication-Subscription

### Availability

This feature is available since openGauss 3.0.0.

### Introduction

Publication-subscription is implemented based on logical replication, with one or more subscribers subscribing to one or more publications on a publisher node. The subscriber pulls data from the publications they subscribe to. Data across database clusters can be synchronized in real time.

### Benefits

The typical application scenarios of publication-subscription are as follows:

- Sending incremental changes in a database or a subset of a database to subscribers as they occur
- Firing triggers when changes reach subscribers
- Consolidating multiple databases into a single one (for example, for analysis purposes)

### Description

Changes on the publisher are sent to the subscriber as they occur in real time. The subscriber applies the published data in the same order as the publisher, so that transactional consistency is guaranteed for publications within a single subscription. This method of data replication is sometimes called transactional replication.

The subscriber database behaves in the same way as any other openGauss instance and can be used as a publisher for other databases by defining its own publications. When the subscriber is treated as read-only by an application, there will be no conflicts in a single subscription. On the other side, conflicts may occur if other write operations are performed by the application or by other subscribers in the same set of tables.

### Enhancements

None.

### Constraints

Publication-subscription is implemented based on logical replication and inherits all restrictions of logical replication. In addition, publication-subscription has the following additional restrictions or missing functions.

- Database schemas and DDL commands are not replicated. Initial schemas can be manually copied by using **pg\_dump --schema-only**. Subsequent schema changes need to be manually synchronized.

- Sequence data is not replicated. The data in serial or identifier columns backed by the sequence in the background will be replicated as part of the table, but the sequence itself will still display the start value on the subscriber. If the subscriber is used as a read-only database, this is usually not a problem. However, if some kind of switchover or failover to the subscriber database is intended, the sequence needs to be updated to the latest value, either by copying the current data from the publisher (perhaps using **pg\_dump**) or by determining a sufficiently large value from the tables themselves.
- Only tables, including partitioned tables, can be replicated. Attempts to replicate other types of relations, such as views, materialized views, or foreign tables, will result in errors.
- Multiple subscriptions in the same database cannot subscribe to the same publication (that is, the same published table). Otherwise, duplicate data or primary key conflicts may occur.
- If a published table contains data types that do not support B-tree or hash indexes (such as the geography types), the table must have a primary key so that UPDATE and DELETE operations can be successfully replicated to the subscription side. Otherwise, the replication will fail, and the message "FATAL: could not identify an equality operator for type xx" will be displayed on the subscription side.

## Dependencies

Publication-subscription depends on the logical replication function.

## 5.17 Foreign Key Lock Enhancement

### Availability

This feature is available since openGauss 3.0.0.

### Introduction

Two types of row locks are added, which are extended from share and update locks to key share, share, no key update, and update locks. A non-primary key update obtains a no key update lock, and a row lock obtained by a foreign key trigger is a key share lock. The two types of locks do not conflict with each other, thereby improving concurrency of foreign key locks.

### Benefits

Most table update operations are non-primary key updates. This feature effectively reduces the blocking of concurrent updates in scenarios with foreign key constraints and improves efficiency.

### Description

When the non-primary key column of a tuple in the parent table is updated, the no key update lock is obtained. When the corresponding tuple in the child table is updated or inserted, the foreign key trigger is triggered to obtain the key share lock of the tuple in the parent table. They do not block each other.

Because row locks that do not conflict with each other are added, multiple transactions are not composed of only share locks. Instead, there are multiple combinations of different row locks according to the following conflict table.

Lock Mode	key share	share	no key update	update
key share				X
share			X	X
no key update		X	X	X
update	X	X	X	X

## Enhancements

None.

## Constraints

- The new row lock does not support the Ustore table.

## Dependencies

None

# 5.18 Data Compression in OLTP Scenarios

## Availability

This feature is available since openGauss 3.0.0.

## Introduction

The feature supports row-store data compression in OLTP scenarios, provides a general compression algorithm, and implements transparent compression of data pages and maintenance of page storage locations to achieve high compression and high performance. Disk persistence is implemented using two types of files: compressed address file (with the file name extension .pca) and compressed data file (with the file name extension .pcd).

## Benefits

Typically, it is applicable where the database disk space needs to be reduced.

## Description

Data compression in OLTP scenarios can reduce the disk storage space of row tables and index data and improve performance in I/O-intensive database systems.

## Constraints

- Only row-store tables and BTree indexes are supported. The feature cannot be used on Ustore and segment-page storage engines.
- Compressed table index files are accessed using mmap. **max\_map\_count** must be set properly based on the number of compressed table files.

# 6 Application Development Interfaces

---

[6.1 Standard SQL](#)

[6.2 Standard Development Interfaces](#)

[6.3 PostgreSQL API Compatibility](#)

[6.4 PL/Java](#)

## 6.1 Standard SQL

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

SQL is a standard computer language used to control the access to databases and manage data in databases. SQL standards are classified into core features and optional features. Most databases do not fully support SQL standards.

openGauss supports most of the core features of SQL:2011 and some optional features, providing a unified SQL interface for users.

### Benefits

All database vendors can use a unified SQL interface, reducing the costs of learning languages and migrating applications.

### Description

For details, see "SQL Reference > SQL Syntax" in the *Developer Guide*.

### Enhancements

None

## Constraints

None

## Dependencies

None

# 6.2 Standard Development Interfaces

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Standard ODBC 3.5 and JDBC 4.0 APIs are supported.

## Benefits

Standard ODBC and JDBC interfaces are provided to ensure quick migration of user services to openGauss.

## Description

Currently, the standard ODBC 3.5 and JDBC 4.0 APIs are supported. The ODBC interface supports SUSE Linux, Windows 32-bit, and Windows 64-bit platforms. The JDBC API supports all platforms.

## Enhancements

The function of connecting JDBC to a third-party log framework is added. JDBC can interconnect with a third-party log framework to meet users' log management and control requirements.

## Constraints

None

## Dependencies

None

# 6.3 PostgreSQL API Compatibility

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

Compatible with PostgreSQL clients and standard APIs.

## Benefits

Compatible with the PostgreSQL clients and standard APIs, and can be seamlessly interconnected with PostgreSQL ecosystem tools.

## Description

Compatible with PostgreSQL clients and standard APIs.

## Enhancements

None

## Constraints

None

## Dependencies

None

# 6.4 PL/Java

## Availability

This feature is available since openGauss 1.0.0.

## Introduction

The Java user-defined field (UDF) is supported.

## Benefits

A development environment is provided for multiple functions.

## Description

With the openGauss PL/Java function, you can choose your favorite Java IDE to write Java methods and install the JAR files containing these methods into openGauss before invoking them. openGauss PL/Java is developed based on open-source tada PL/Java 1.5.2. PL/Java uses Huawei JDK V100R001C00SPC190B003-b09.

## Enhancements

None



## Constraints

- Java UDFs can implement simple Java computing. However, do not encapsulate services in Java UDFs.
- Do not connect to a database in any way (for example, JDBC) in Java functions.
- Use Huawei JDK V100R001C00SPC190B003-b09 to compile Java methods and JAR files.
- Currently, only data types listed in [Table 6-1](#) are supported. Other data types, such as user-defined data types and complex data types (for example, Java array and its derived types) are not supported.
- Currently, UDAF and UDTF are not supported.

**Table 6-1** PL/Java mapping for default data types

openGauss	Java
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

## Dependencies

PL/Java depends on the Java Development Kit (JDK) environment. Currently, JDK is included in openGauss and installation is not required. If you have installed the

same or different versions of JDK, no conflict will occur. openGauss uses Huawei JDK V100R001C00SPC190B003-b09 to run PL/Java.

# 7 AI Capabilities

---

The history of artificial intelligence (AI) can be dated back to as early as the 1950s, even longer than the history of the database development. However, the AI technology has not been applied on a large scale for a long time due to various objective factors, and even experienced several obvious troughs. With the further development of information technologies in recent years, factors that restrict the AI development have been gradually weakened, and the AI, big data, and cloud computing (ABC) technologies are born.

The combination of AI and databases has been a trending research topic in the industry in recent years. openGauss has participated in the exploration of this domain earlier and achieved phased achievements. An AI submodule DBMind is provided for the database. Compared with other functions, it is more independent. This module can be divided into AI4DB, DB4AI, and AI in DB.

- AI4DB uses AI technologies to optimize database execution performance as well as achieve autonomy and O&M free. It includes self-tuning, self-diagnosis, self-security, self-O&M, and self-healing.
- DB4AI streamlines the E2E process from databases to AI applications, drives AI tasks through databases, and unifies the AI technology stack to achieve out-of-the-box, high performance, and cost saving. For example, SQL-like statements are used to implement functions such as recommendation system, image retrieval, and time series forecast. The advantages of high parallelism and column store of databases can be fully utilized to avoid the cost of data and fragmented storage and avoid security risks caused by information leakage.
- AI in DB modifies the database kernel to implement functions that cannot be implemented in the original database architecture. For example, AI algorithms are used to improve the database optimizer to implement more accurate cost estimation.

The functions described in this section are stored in the **bin/dbmind** directory of the database installation directory (*\$GAUSSHOME*). The sub-functions are stored in the **components** subdirectory of **bin/dbmind**. To invoke DBMind, you can run the **gs\_dbmind** command. In addition, the built-in AI functions (such as DB4AI) of the database are presented in the form of SQL syntaxes and system functions.

## [7.1 AI4DB: Autonomous Database O&M](#)

## [7.2 DB4AI: Database-driven AI](#)

## 7.3 AI in DB

# 7.1 AI4DB: Autonomous Database O&M

## 7.1.1 Database Metric Collection, Forecast, and Exception Detection

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

This feature serves as an AI tool integrated into openGauss and can be used to collect and forecast database metrics, as well as monitor and diagnose exceptions. It is a component in the DBMind suite. Currently, this feature is compatible with the Prometheus platform to collect database system metrics. Prometheus exporters are provided to collect and process database monitoring metrics. By monitoring the time series data of metrics, you can forecast the future load trend and diagnose problems.

### Benefits

- This feature greatly simplifies the work of O&M personnel, releases a large number of labor resources, and reduces costs for the company.
- You can use the metric collection, monitoring, and forecast functions to detect problems in advance, preventing database exceptions from causing greater loss.

### Description

Prometheus is a popular open-source monitoring system in the industry. It is also a time series database. The collector of Prometheus is called exporter, which is used to collect metrics of monitored modules. To interconnect with the Prometheus platform, DBMind provides two types of exporters: openGauss-exporter for collecting database metrics and reprocessing-exporter for reprocessing the collected metrics.

This feature supports forecast of collected metrics. You can specify key performance indicators (KPIs) to be forecasted by modifying configuration files. This helps you find metric trends and perform O&M operations in a timely manner. For example, you can forecast the memory usage to detect memory leakage and forecast the disk usage to expand the capacity at a proper time.

### Enhancements

This feature is greatly improved in openGauss 3.0.0 and is compatible with the Prometheus platform. Two exporters are used to connect to Prometheus.

## Constraints

- The database is normal, and the data directory has been written into environment variables.
- The Python version must be 3.6 or later.
- The Prometheus monitoring platform is configured and the Prometheus service is started so that monitoring data can be collected.

## Dependencies

Prometheus

## 7.1.2 Root Cause Analysis for Slow SQL Statements

### Availability

This feature is available since openGauss 3.0.0.

### Introduction

Slow SQL statements have always been a pain point in data O&M. How to effectively diagnose the root causes of slow SQL statements is a big challenge. Based on the characteristics of openGauss and the slow SQL statement diagnosis experience of DBAs on the live network, this tool supports more than 15 root causes of slow SQL statements, outputs multiple root causes based on the possibility, and provides specific solutions.

### Benefits

This feature provides customers with fast and reliable slow SQL statement discovery and root cause analysis functions, greatly simplifying the work of O&M personnel.

### Description

Based on the Prometheus data collection solution, data required for root cause analysis for slow SQL statements is collected, including system resource information (CPU usage, memory usage, and I/O), load information (QPS), large process information (including external large processes and scheduled database tasks), slow SQL statement text information, start time and end time of slow SQL statement execution, slow SQL statement execution plan, temporary file information, and so on. Then, this feature calculates the most matched root cause of slow SQL statements based on the AI algorithm, and provides suggestions and confidence.

### Enhancements

None.

### Constraints

- The database is normal, and the client can be connected properly.

- An environment running Python 3.6 or later is available.
- The information about slow SQL statements is obtained from the workload diagnosis report (WDR). In the database WDR, slow SQL statements are marked. The GUC parameter **track\_stmt\_stat\_level** is enabled by default. Otherwise, you need to manually enable it. Generally, the **track\_stmt\_stat\_level** is set to 'off, L0'. Higher levels will affect the performance. Data collection is implemented by the Prometheus solution. Therefore, you need to configure the Prometheus data collection platform. This feature focuses on algorithms and obtains metric sequence information from Prometheus.

## Dependencies

None.

## 7.1.3 Index Recommendation

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

This feature serves as an intelligent database index recommendation tool that covers multiple task levels and application scenarios. It provides the single-query index recommendation function, virtual index function, and workload-level index recommendation function to provide reliable index recommendations for users.

### Benefits

This feature provides the quick and reliable index recommendation function, greatly simplifying the work of O&M personnel.

### Description

The single-query index recommendation function allows users to directly perform operations in the database. This feature generates recommended indexes for a single query statement entered by users based on the semantic information of the query statement and the statistics of the database. The virtual index function allows users to directly perform operations in the database. This feature simulates the creation of a real index to avoid the time and space overhead required for creating a real index. Based on the virtual index, users can evaluate the impact of the index on the specified query statement by using the optimizer. The workload-level index recommendation can be used by running scripts outside the database. This feature uses the workload of multiple DML statements as the input to generate a batch of indexes that can optimize the overall workload execution performance.

### Enhancements

None.

## Constraints

The database is normal, and the client can be connected properly.

The gsql tool has been installed by the current user, and the tool path has been added to the *PATH* environment variable.

An environment running Python 3.6 or later is available.

## Dependencies

None.

## 7.1.4 Parameter Tuning and Diagnosis

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

This feature serves as a parameter tuning tool integrated into databases. It uses AI technologies such as deep reinforcement learning and global search algorithms to obtain the optimal database parameter settings without manual intervention. It is not forcibly deployed with the database environment. It can be independently deployed and run without the database installation environment.

### Benefits

This tool can quickly provide the parameter adjustment configuration of the current load in any scenario, reducing database administrator's manual intervention, improving the O&M effect, and meeting customer expectations.

### Description

The tuning program can run in any of the following modes:

- **recommend**: Log in to the database using the specified username, obtain the feature information about the running workload, and generate a parameter recommendation report based on the feature information. Report improper parameter settings and potential risks in the current database. Output the currently running workload behavior and characteristics. Output the recommended parameter settings. In this mode, the database does not need to be restarted. In other modes, the database may need to be restarted repeatedly.
- **train**: Modify parameters and execute the benchmark based on the benchmark information provided by users. The reinforcement learning model is trained through repeated iteration so that you can load the model in **tune** mode for optimization.
- **tune**: Use an optimization algorithm to tune database parameters. Currently, two types of algorithms are supported: deep reinforcement learning and global search algorithm (global optimization algorithm). The deep reinforcement learning mode requires **train** mode to generate the optimized

model after training. However, the global search algorithm does not need to be trained in advance and can be directly used for search and optimization.

## Enhancements

None.

## Constraints

- The database is normal, the client can be properly connected, and data can be imported to the database. As a result, the optimization program can perform the benchmark test for optimization effect.
- To use this tool, you need to specify the user who logs in to the database. The user who logs in to the database must have sufficient permissions to obtain sufficient database status information.
- If you log in to the database host as a Linux user, add `$GAUSSHOME/bin` to the `PATH` environment variable so that you can directly run database O&M tools, such as `gsql`, `gs_guc`, and `gs_ctl`.
- The recommended Python version is Python 3.6 or later. The required dependency has been installed in the operating environment, and the optimization program can be started properly. You can install a Python 3.6+ environment independently without setting it as a global environment variable. You are not advised to install the tool as the root user. If you install the tool as the root user and run the tool as another user, ensure that you have the read permission on the configuration file.
- This tool can run in three modes. In **tune** and **train** modes, you need to configure the benchmark running environment and import data. This tool will iteratively run the benchmark to check whether the performance is improved after the parameters are modified.
- In **recommend** mode, you are advised to run the command when the database is executing the workload to obtain more accurate real-time workload information.
- By default, this tool provides benchmark running script samples of TPC-C, TPC-H, TPC-DS, and sysbench. If you use the benchmarks to perform pressure tests on the database system, you can modify or configure the preceding configuration files. To adapt to your own service scenarios, you need to compile the script file that drives your customized benchmark based on the **template.py** file in the **benchmark** directory.

## Dependencies

None.

## 7.1.5 Slow SQL Statement Discovery

### Availability

This feature is available since openGauss 1.1.0.

### Introduction

This feature serves as an SQL statement execution time forecast tool. It forecasts the execution time of SQL statements based on the statement logic similarity and



historical execution records without obtaining the SQL statement execution plan using a template.

## Benefits

- This feature does not require users to provide SQL execution plans. Therefore, the database performance is not affected.
- Different from other algorithms in the industry that are limited to OLAP or OLTP, this feature is more widely used.

## Description

The SQLdiag focuses on the historical SQL statements of the database, summarizes the execution performance of the historical SQL statements, and then uses the historical SQL statements to infer unknown services. The execution duration of SQL statements in the database does not differ greatly in a short period of time. SQLdiag can detect the statement result set similar to the executed SQL statements from historical data and predict the execution duration of SQL statements based on the SQL vectorization technology and template-based method.

## Enhancements

None.

## Constraints

- The historical logs and the format of the workload to be predicted meet the requirements. You can use the GUC parameter of the database to enable the collection or use the monitoring tool to collect logs.
- To ensure the prediction accuracy, the historical statement logs provided by users should be as comprehensive and representative as possible.
- The Python environment has been configured as required.

## Dependencies

None.

# 7.2 DB4AI: Database-driven AI

## Availability

This feature is available since openGauss 2.1.0.

## Introduction

DB4AI uses database capabilities to drive AI tasks and implement data storage and technology stack isomorphism. By integrating AI algorithms into the database, openGauss supports the native AI computing engine, model management, AI operators, and native AI execution plan, providing users with inclusive AI technologies. Different from the traditional AI modeling process, DB4AI one-stop

modeling eliminates repeated data flowing among different platforms, simplifies the development process, and plans the optimal execution path through the database, so that developers can focus on the tuning of specific services and models. It outcompetes similar products in ease-of-use and performance.

## Benefits

- With this feature, you do not need to manually compile AI model code. Instead, you can use out-of-the-box SQL statements to train and forecast machine learning models, reducing the learning and usage costs.
- Extra overhead that is caused by fragmented data storage and repeated data migration can be avoided.
- A higher execution efficiency can be achieved. With this feature, the AI model training efficiency is high. Compared with manual model training, the performance is improved by several times.
- Stricter security protection prevents data leakage during AI model training.

## Description

openGauss supports the native DB4AI capability. By introducing native AI operators, openGauss simplifies the operation process and fully utilizes the optimization and execution capabilities of the database optimizer and executor to obtain the high-performance model training capability in the database. With a simpler model training and forecast process and higher performance, developers can focus on model tuning and data analysis in a shorter period of time, avoiding fragmented technology stacks and redundant code implementation.

## Enhancements

More algorithms are supported in openGauss 3.0.0.

## Constraints

- The database is running properly.

## Dependencies

None.

# 7.3 AI in DB

## 7.3.1 Predictor: AI Query Time Forecasting

### Availability

This feature is available since openGauss 1.0.0.

### Introduction

Predictor is a query time forecasting tool that leverages machine learning and has online learning capability. Predictor can predict the execution time of a plan in the

database kernel by continuously learning the historical execution information collected in the database.

## Benefits

This feature predicts the SQL statement execution time based on the AI model before the SQL statement is executed. Based on the estimated SQL time, you can detect possible abnormal SQL statements and optimize or schedule them in advance to prevent system running from being affected.

## Description

The prediction of AI query time depends on the collection of local query plans. During query execution, you need to collect the actual query plan (including the plan structure, operator type, related data source, and filter criteria), actual execution time of each operator node, cost estimated by the optimizer, number of rows returned by the optimizer, number of rows estimated by the optimizer, and number of parallel rows. These records are stored in data tables and managed persistently, and expired data will be cleared periodically.

After the database kernel automatically collects historical data, the administrator encodes the data and sends a request to Python through the CURL API to configure the model, send data, and trigger training. You can call the TensorBoard API to monitor the training process. The model returns the prediction accuracy of each project and saves the final model. The database updates the system tables related to the model information based on the result returned by the AI engine.

This feature is triggered when **explain (analyze on, predictor <model\_name>) SELECT... "** is executed and the model is converged. The database determines whether the current model meets the requirements based on the plan encoding. If the model meets the requirements, the model loading request is sent to Python (the model needs to be loaded only once after the model training is complete). After the model is loaded, the plan encoding file is inferred and the prediction result of each node is returned.

## Enhancements

None.

## Constraints

- The database system is normal. The user successfully logs in to the database through identity authentication and accesses the authorized data.
- The SQL syntax is correct and no error is reported.
- SQL statements executed by users do not cause database exceptions.
- In the historical performance data window, the cluster concurrency is stable, the cluster scale, table structure, and table quantity remain unchanged, the data volume does not change abruptly, and the GUC parameters related to query performance remain unchanged. If the preceding conditions are damaged, the model becomes invalid. In this case, you need to invalidate all historical performance data, collect data again, and retrain the model.

- When a model is loaded, the structure, parameters, and training information of the original model are completely saved. If the original model information is lost, the model cannot be loaded and needs to be trained again.
- Historical performance data can be properly collected and encoded, and no error is reported when the encoded data is properly parsed.
- You can install the following software by using the provided installation scripts or by yourself: Python==3.6.4, configparser==3.8.1, Flask==0.12.2, Keras==2.2.4, numpy==1.16.4, scikit-learn==0.19.1, pandas==0.25.1, tensorboard==1.14.0, and tensorflow-gpu==1.14.0 or tensorflow==1.14.0.
- OpenSSL has been installed in the user environment, and the certificate has been generated using a script or based on the usage description.
- Currently, administrators need to manually synchronize models across database namespaces. Automatic synchronization is not supported.

## Dependencies

None.

# 8 Tool Chain

---

[8.1 MySQL to openGauss Migration Tool Chameleon](#)

[8.2 OpenGauss Client Tool DataStudio](#)

## 8.1 MySQL to openGauss Migration Tool Chameleon

### Availability

This feature is available since openGauss 3.0.0.

### Introduction

Chameleon is Python-based. It supports real-time data replication from MySQL to openGauss. The tool can replicate initial full data and incremental data in real time to migrate them from MySQL to openGauss. It supports migration of various data types in MySQL, and maintains the precision of floating-point data, including decimal, dec, numeric, float, float4, float8, real, double, double precision and fixed after migration.

### Benefits

You can use Chameleon to migrate data from the MySQL database to the openGauss database.

### Description

Chameleon provides full and incremental data replication functions to migrate data from the MySQL database to the openGauss database. Chameleon stores the mappings between MySQL and openGauss data types and supports migration of various data types in MySQL. Specifically, floating-point data in MySQL, including decimal, dec, numeric, float, float4, float8, real, double, double precision and fixed, will be converted to the numeric[p, s] type in openGauss if the precision is specified or contained by default. If the precision is not specified, the data will be converted to the numeric data type in openGauss. In this way, the data precision remains unchanged after offline and online migration.

## Enhancements

None.

## Constraints

- MySQL 5.7 is supported.
- If incompatibility still exists after data type mapping, table data migration fails, but the subsequent offline data migration process does not terminate.

## Dependencies

None.

# 8.2 OpenGauss Client Tool DataStudio

## Availability

You can obtain the installation package of DataStudio from the openGauss official website or obtain the source code from Gitee and compile by yourself.

## Introduction

DataStudio is a universal and integrated development environment for developers and database administrators. It simplifies the development and management of openGauss databases.

DataStudio allows database developers to:

- Manage and create database objects.
- Run SQL statements or SQL scripts.
- Edit and run PL/SQL statements.
- Import and export table data.

## Benefits

As a client software used with openGauss databases (currently, only openGauss is supported), DataStudio helps database developers easily develop applications and provides key database features on the GUI. The tool makes database inclusive and improves user loyalty.

## Description

Start DataStudio on Windows and you can remotely connect to openGauss and perform the following operations:

- Connect to the openGauss database over an integrated GUI-based development environment.
- Efficiently develop SQL.

- Manage or create database objects (databases, schemas, functions, stored procedures, tables, sequences, columns, indexes, constraints, views, users, roles, and tablespaces).
- Run SQL statements or SQL scripts.
- Create and execute a stored procedure.
- Add, delete, modify, and query table data.
- Import and export table data.
- Display and export DDL data.
- Import and export connection information.
- Format SQL statements.
- View SQL execution history.
- Display the execution plan and ER diagram.

## Enhancements

None.

## Constraints

- Supports Windows OS.
- Supports JDK 11 or later versions.

## Dependencies

Currently, only openGauss is supported.

# 9 Middleware

---

[9.1 Distributed Database Capability](#)

[9.2 Deploying a Distributed Database Using Kubernetes](#)

## 9.1 Distributed Database Capability

### Availability

This feature is available since openGauss 2.1.0.

### Introduction

This feature uses the distributed middleware shardingSphere to provide openGauss the distributed database capability. When 16 Kunpeng 920 (128 cores) nodes are used for networking (1 x shardingSphere-proxy, 7 x shardingSphere-jdbc, 8 x openGauss), the perfect sharding performance is greater than 10 million transactions per minute C (tpmC).

### Benefits

A distributed database that logically has no resource restriction can be built over the middleware.

### Description

With the sharding capability of shardingSphere, multiple openGauss databases can logically form a larger database with distributed transactions and elastic scaling capabilities. The usage method is the same as that of an openGauss database.

### Enhancements

None.

### Constraints

None.



## Dependencies

Shardingsphere middleware

# 9.2 Deploying a Distributed Database Using Kubernetes

## Availability

This feature is available since openGauss 2.1.0.

## Introduction

Deploys a distributed database in one-click mode.

## Benefits

Quickly builds a distributed database, and verifies and uses the distributed capability.

## Description

Patroni is used to implement planned switchover and automatic failover in case of faults. HAProxy is used to implement read and write load balancing between the primary and standby openGauss nodes. ShardingSphere is used to implement distributed capabilities. All functions are packaged into images and one-click deployment scripts are provided.

## Enhancements

None.

## Constraints

Only CentOS and openEuler are supported.

## Dependencies

ShardingSphere, Patroni, HAProxy