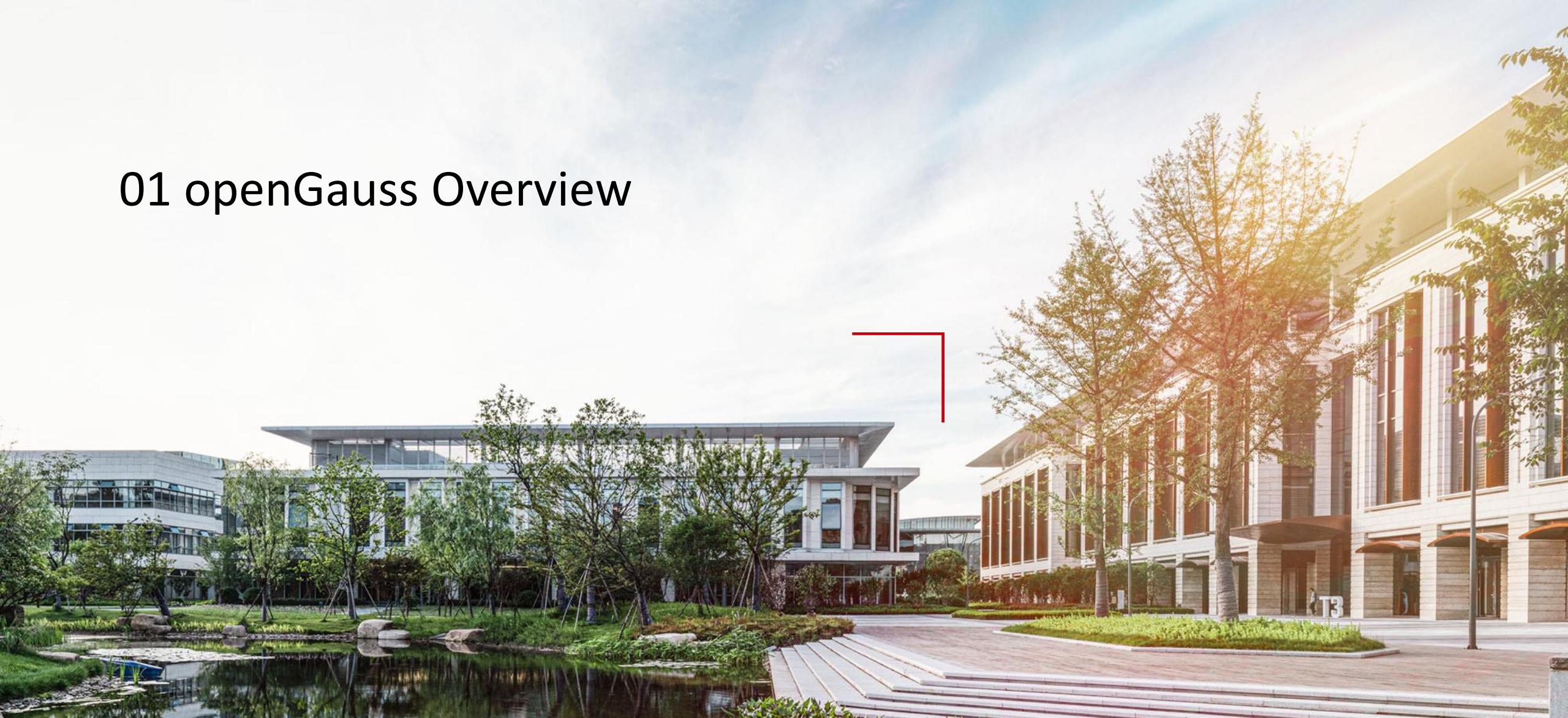


01 openGauss Overview



Foreword

- Database emerged in the 1960s as a new discipline for automating information management, and has since become an important branch of computer science. Data processing and database technology have found their way into more and more computer applications as such applications have developed. And while a database is a product of data management, data management is the core task involving a database. It involves data classification, organization, encoding, storage, retrieval, and maintenance. This section describes the development and features of the openGauss database.

Objectives

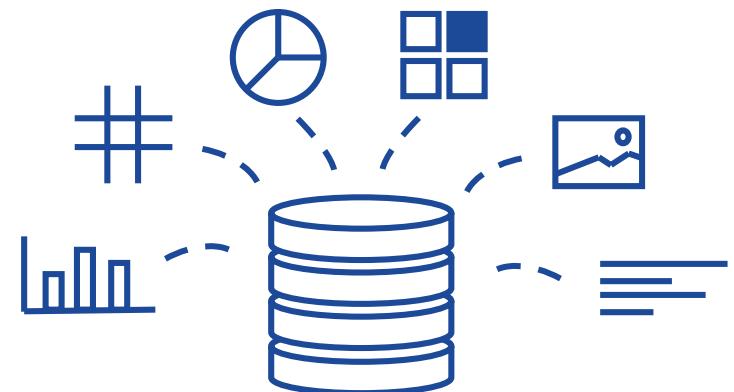
- Upon completion of this course, you will be able to:
 - Understand what a database is, as well as the history behind the development of the technology.
 - Be familiar with how the architecture of relational databases has evolved.
 - Understand the main applications for relational databases.
 - Understand where the openGauss database is positioned, and the history of its development.
 - Understand the technical specifications and basic functions of openGauss.

Contents

- 1. Database Overview**
2. Introduction to openGauss
3. openGauss Technical Specifications
4. Basic Functions

What Is a Database?

- Data is facts or statistics used to represent something for observation or analysis.
- Data can represent carrier of information as symbols, text, numbers, voices, images, or videos. Data and information are inseparable and data is basically a form of information. Data itself is meaningless. Data becomes information only when it is put to use.
- Data in a computer system is expressed as various letters, digits, sounds, graphs, images, and more, which are formed through binary units "1" and "0". Binary is converted into more conventional forms of data with processing.
- Forms
 - **Digital data:** data consisting of discrete symbols, characters, and numerals. For example, statistics or measurement data stored on a computer.
 - **Analog data:** refers to a continuous range of values over an interval. This data takes a physical form, such as video, image, text, or audio.



Database

A database is an, often large, collection of organized data that is stored in a computer, often for a long time, and with the ability to be accessed. Data in a database is organized, described, and stored according to a data model. The data in a database is efficient, relatively independent, and scalable, and can be accessed by various users.



Permanent

Organized

Sharable

DBMS

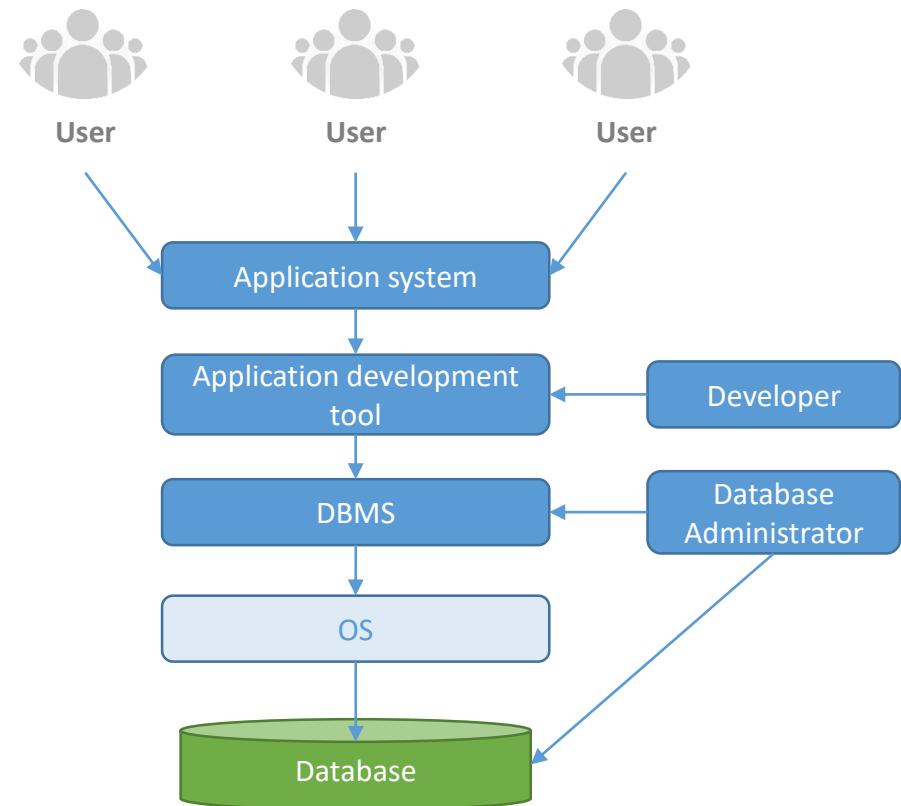
- A database management system (DBMS) consists of a collection of interconnected data and a set of programs used to access the data. This dataset is usually called a database. The main purpose of DBMS is to provide a convenient and efficient way to access database information.
- A DBMS acts as an interface between a database and its users or programs, allowing users to retrieve, update, and manage information in an organized and optimized manner. In addition, the DBMS helps monitor and control the database and provides various management operations, such as performance monitoring, optimization, backup, and recovery.
- Typical database management systems include Oracle, Microsoft SQL Server, Access, MySQL, and PostgreSQL.

DBMS - Functions

- **Data definition:** The DBMS provides a data definition language (DDL), through which users can easily define data objects in a database.
- **Data manipulation:** The DBMS also provides a data manipulation language (DML), which can be used by users to manipulate data and perform basic operations on the database, such as query, insert, delete, and modify.
- **Database operation management:** The DBMS manages and controls the creation, operation, and maintenance of databases in a unified manner to ensure data security and integrity, concurrent use of data by multiple users, and to ensure the system recovers after a fault.
- **Interfaces and tools for convenient and effective database access:** Programmers can develop database applications through an interface. Database administrators (DBAs) can use tools to manage databases.
- **Database creation and maintenance:** include the input and conversion of initial data to the database, the dump and restoration of the database, the reorganization of the database, and the performance monitoring and analysis. These functions are usually implemented in software.

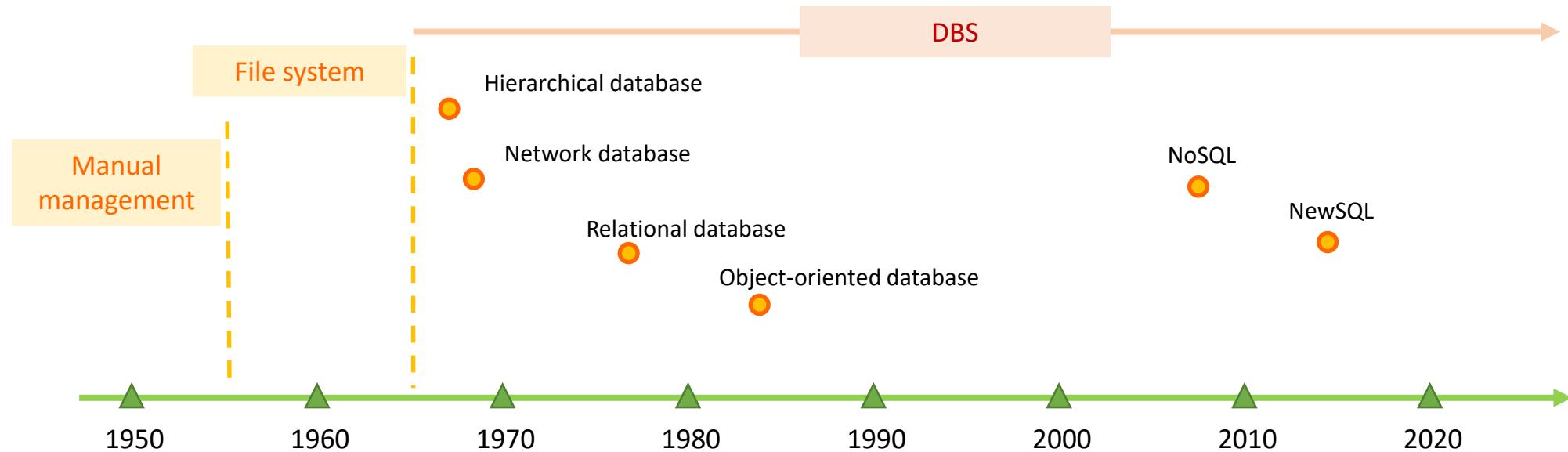
DBS

- A database system (DBS) includes hardware and software. Hardware is there as storage through a medium such as a hard drive. It is also there to support the software, which mainly includes the DBMS, the operating system (OS) that supports the operation of the DBS, and the access technology that supports the application development in multiple languages.
- The DBS is the sum of all the database components. A complete DBS consists of the database, DBMS, application development tool, application system, database administrator, and users.



Three Stages of Development: Data Management Technologies

Data management is to classify, organize, encode, query, and maintain various forms of data. It mainly goes through three stages: manual management, file system, and DBS, each of which has evolved towards reducing data redundancy, enhancing data independence, and facilitating data operations.



Comparison of Data Management at Different Stages

- Each stage of data management has its own background and features, and the data management technology is continuously improving in each stage. The following table compares data management at the three stages.

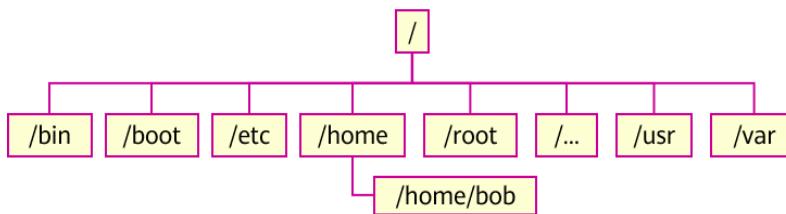
Stage	Manual Management (Mid 1950s)	File System (Late 1950s to Mid 1960s)	DBS (Late 1960s)
Application	Scientific computing	Scientific computing and management	Management of large-scale data and distributed data
Hardware	No direct access storage devices	Tapes, disks, and magnetic drums	Large-capacity disks, erasable CD-ROMs, and tape drives supporting on-demand capacity expansion
Software	No dedicated management software	File system	DBS
Data Processing Method	Batch processing	Online real-time processing and batch processing	Online real-time processing, batch processing, and distributed processing
Data Manager	User/Program	File system	DBMS
Data Application and Extension	Oriented to a certain application, difficult to expand	Oriented to a certain application system, difficult to expand	Oriented to multiple application systems, easy to expand
Data Sharing	No sharing and high redundancy	Poor sharing and high redundancy	Good sharing and low redundancy
Data Independence	Poor data independence	Good physical independence but poor logical independence	High physical independence and good logical independence
Data Structuring	Unstructured data	Structured only within the record, but unstructured on the whole	Unified data model and overall structuring
Data Security	Application protection	File system protection	Comprehensive security protection

Why Do We Use Databases?

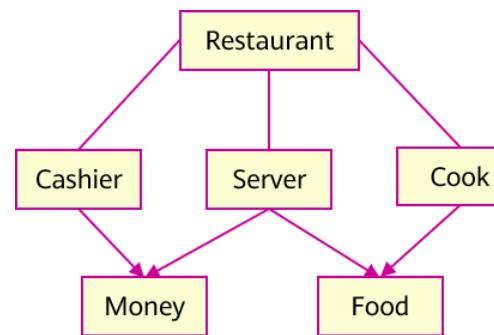
- A database can store data efficiently and orderly, enabling us to manage data more quickly and conveniently:
 - Databases can store a large amount of information, and therefore data, in a structured manner, facilitating effective retrieval and access by users.
 - The database can effectively maintain the consistency and integrity of data information and reduce data redundancy.
 - The database can meet the sharing and security requirements of applications. In most cases, data is stored in databases for security purposes.
 - The database technology can be easily and intelligently analyzed to produce new and useful information.

Database Models

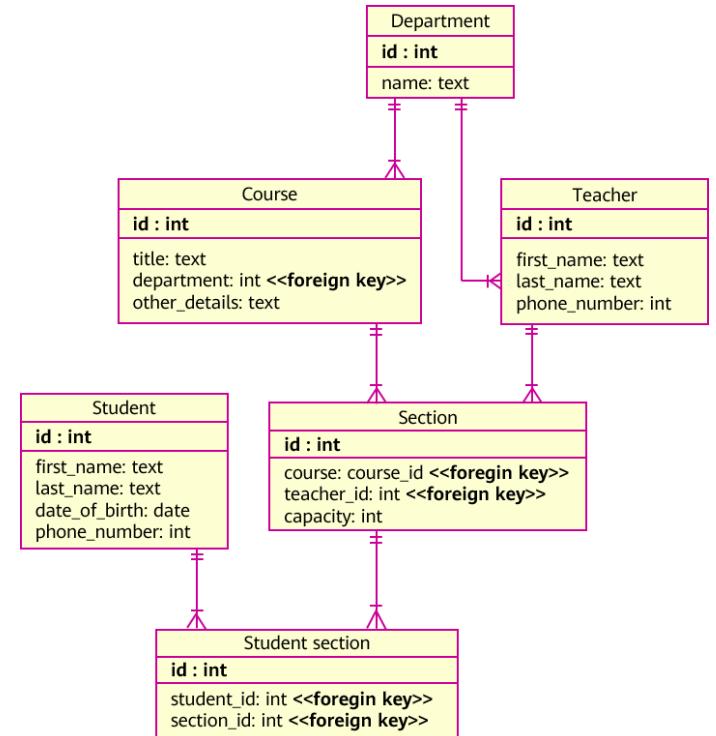
- Since the middle of the 20th century, many interesting database models, such as **hierarchical model**, **network model**, and **relational model**, have emerged. Some database models are no longer used, while others are still playing a significant role.



Hierarchical model



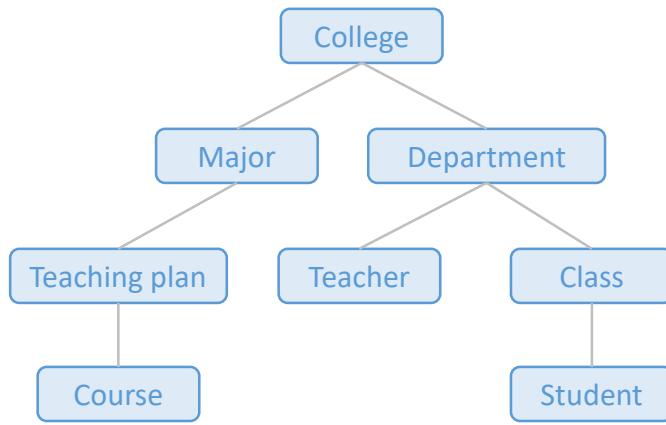
Network model



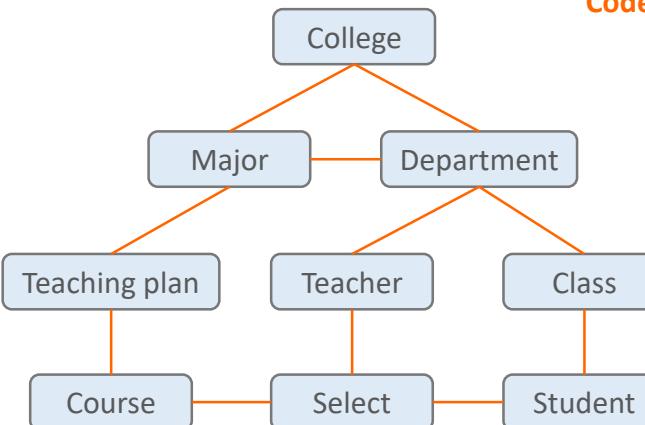
Relational model

Database Models: Hierarchical, Network, and Relational Models

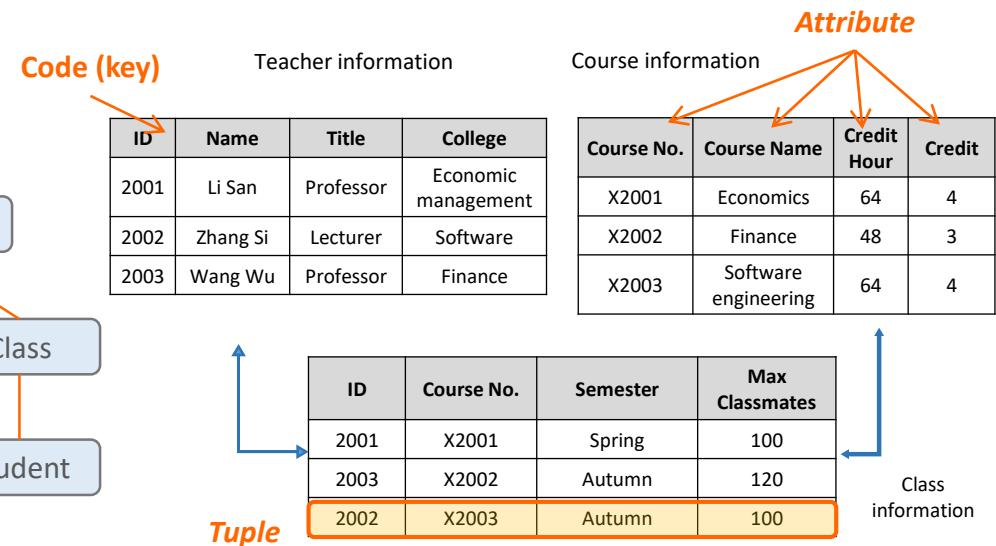
Hierarchical model



Network model



Relational model



- Hierarchical model

- If there is only one node that has no parent, the node is called "root".
- All nodes except the root node have only one parent node.

- Network model

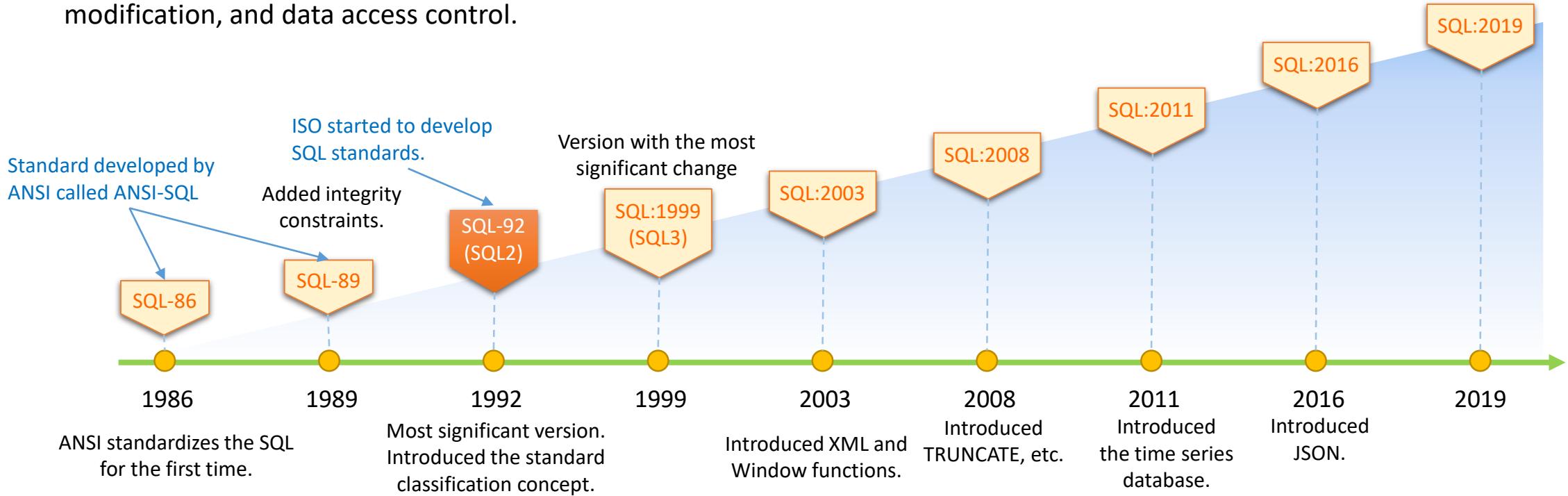
- Allows more than one node to have no parent.
- A node can have more than one parent.

- Relational model

- It is based on a rigorous data concept.
- Relationships must be normalized.
- The component of a relationship must be an indivisible data item.

SQL

- Structured Query Language (SQL) is the standard language for relational databases. SQL is a general-purpose and powerful relational database language. It is a standard interface for accessing relational data and is also the basis for interoperation between different database systems.
- SQL is based on relational algebra and tuple relationship calculations. It integrates data query, data operations, data definition, and data control functions. The scope of SQL includes data insertion, query, update, and deletion, database schema creation and modification, and data access control.

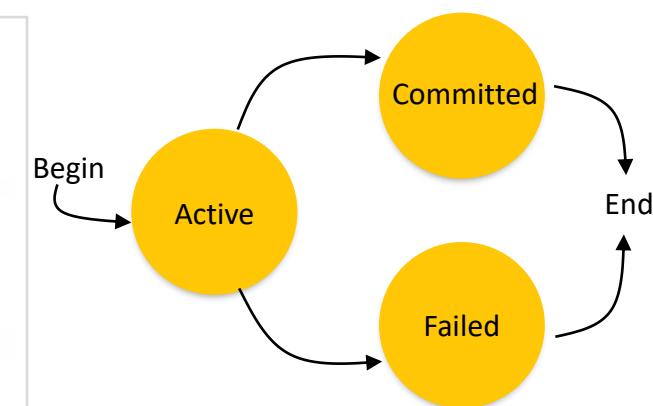


Transaction

- A transaction is a logical unit of program execution, consisting of a series of operations for accessing or updating data in the system. In computer terminology, a transaction usually refers to a database transaction.
- A database transaction usually contains a sequence of read/write operations on the database. Transactions in a database environment have two main purposes:
 - 1. To provide a method for restoring the database operation sequence from a failure while keeping database consistency even in case of system failure.
 - 2. To provide isolation between programs accessing a database concurrently, so that they do not interfere with one another.
- Transaction status:
 - A transaction is atomic and is regarded as a whole. That is, it can only be in one of the following three states at any time: active, committed, and failed.

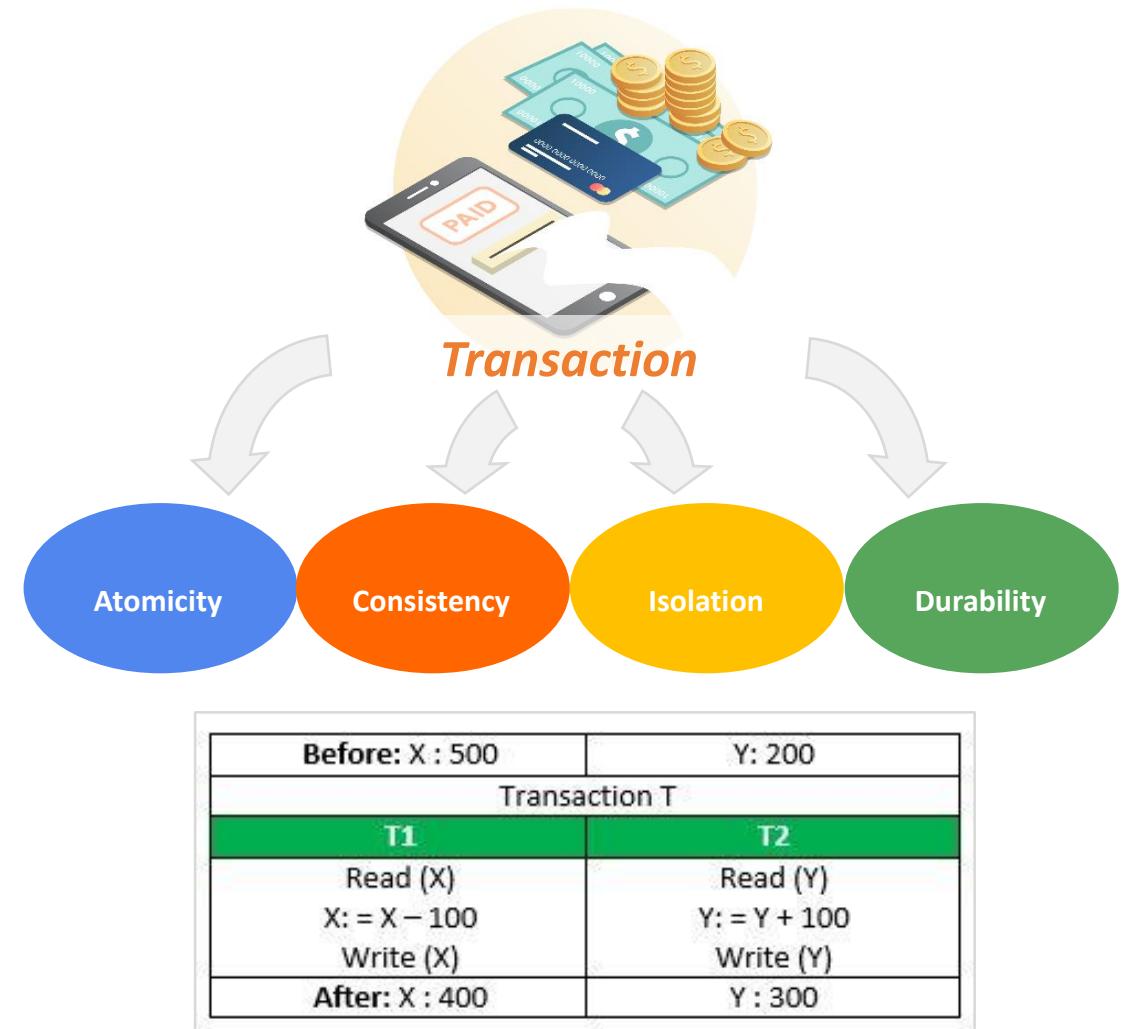
One transaction

```
BEGIN;  
UPDATE accounts SET balance = balance - 100.00  
WHERE name = 'Alice';  
UPDATE branches SET balance = balance - 100.00  
WHERE name = (SELECT branch_name FROM accounts  
WHERE name = 'Alice');  
UPDATE accounts SET balance = balance + 100.00  
WHERE name = 'Bob';  
UPDATE branches SET balance = balance + 100.00  
WHERE name = (SELECT branch_name FROM accounts  
WHERE name = 'Bob');  
COMMIT;
```

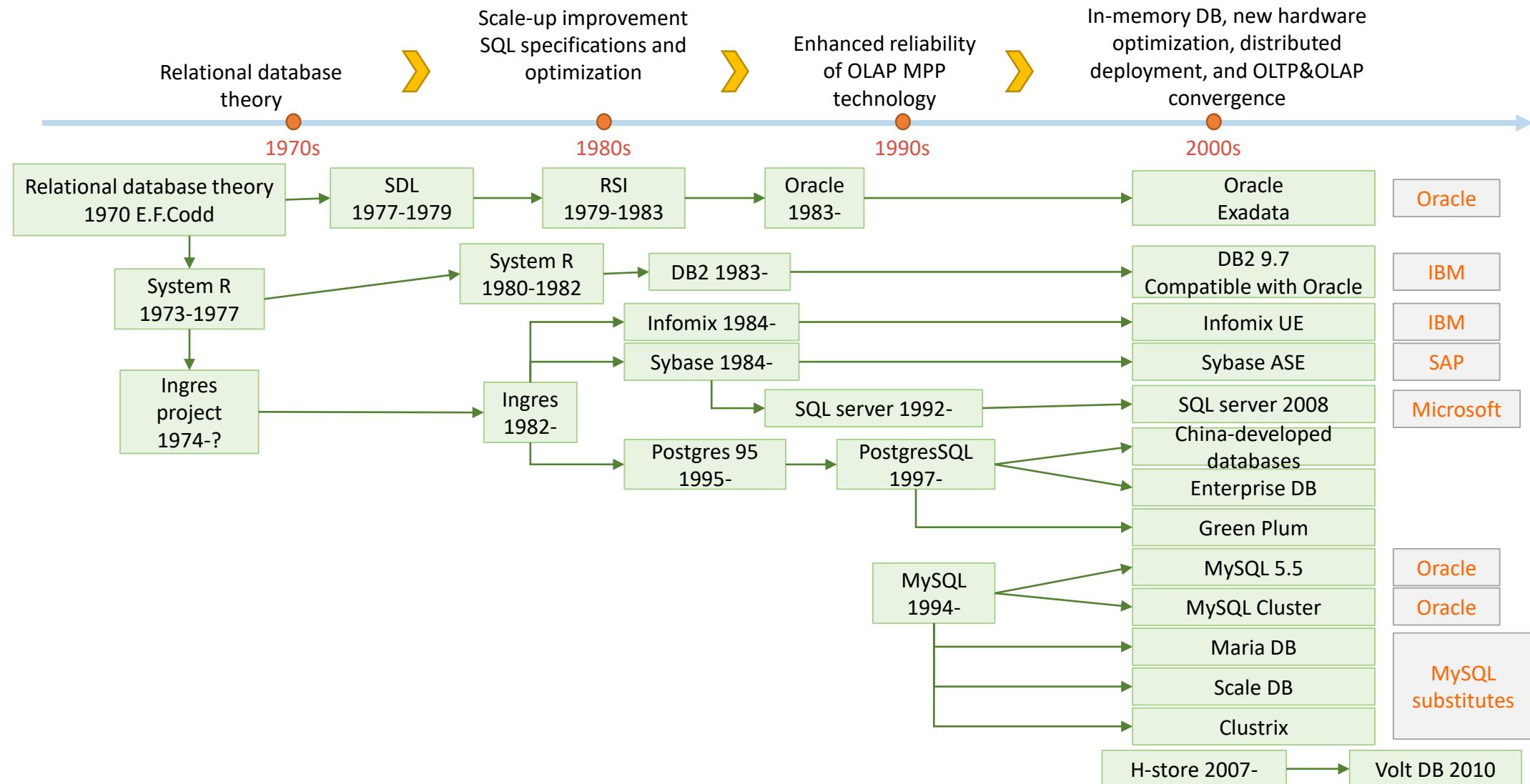


ACID Feature of Relational Databases

- **Atomicity**
 - A transaction is the logical working unit of a database. Atomicity means that you guarantee that either all operations in the transaction succeed or none do.
- **Consistency**
 - This ensures that a transaction can only transit from one consistent state to another consistent state.
- **Isolation**
 - This ensures that transactions do not interfere one another. That is, the internal operations and data used in a transaction are isolated from other transactions, and concurrent transactions do not affect each other.
- **Durability**
 - Once a transaction is committed, changes made to data in the database are permanent. Operations or faults occurring after the commit do not affect the result of the transaction.



The History of Relational Database Development



Database Challenges

The databases of large enterprises need to support very complex searches, with users expecting responses almost in real time. As such, database administrators often need to adopt various methods to help enterprises improve performance. Some of the common challenges they face include:

- **Sharp increase in data volume.** The explosive growth of data from sensors, networked devices, and many other sources has kept database administrators busy with data management and organization.
- **Ensuring data security.** Nowadays, data is everywhere posing great leakage risks, and hackers' attack methods are emerging. Therefore, it is more important than ever to ensure data security while ensuring that users can easily access data.
- **Meeting ever-changing requirements.** In today's fast-growing business environment, enterprises need to access data in real time to make the best decisions in a timely manner and seize new opportunities.
- **Managing and maintaining databases and infrastructure.** Database administrators need to continuously monitor problems in the database and perform preventive maintenance, application software upgrades, and patch installations. As databases become increasingly complex and data volumes increase, enterprises need to recruit more personnel to monitor and optimize databases, increasing costs.
- **Breaking the scalability limitation.** In order to survive, modern enterprises must continue to develop, and the corresponding data management must keep pace with the times. However, it is often difficult for database administrators to predict the future data volume of an enterprise, especially if the database is deployed locally.

NoSQL

- NoSQL, short for Not Only SQL, is a DBMS different from traditional relational databases. NoSQL is used to store ultra-large-scale data. These types of databases do not require fixed storage modes and can be scaled out without extra operations. Based on the structuring method and application scenario, the databases are classified into the following types:
- **Key-value database oriented to high-performance concurrent read/write**
 - Key-value databases feature high concurrent read/write performance. A key-value database is a database that stores data using key-value pairs. It is similar to the map in Java. The entire database can be regarded as a large map, and each key corresponds to a unique value.
- **Document database oriented to massive data access**
 - The main feature of this database is that data can be quickly queried from a large amount of data. Document storage typically uses an internal notation that can be processed directly in the application, primarily in JSON. JSON documents can also be stored as plain text in a key-value store or in a relational DBS.
- **Search engine for content searching**
 - The search engine is a NoSQL DBS that is used to search for data content. It is mainly used for quasi-realtime processing and analysis of massive data and can be used for machine learning and data mining.
- **Distributed database oriented to scalability**

NoSQL – CAP Theory

- The basic demand of NoSQL is to support distributed storage. Strict consistency and availability need to be balanced.
- CAP theory: A distributed system cannot meet consistency (C), availability (A), and partition fault tolerance (P) requirements at the same time, and can meet at most two of them. Partition fault tolerance is a basic requirement for a distributed system. Otherwise, the system cannot be called a distributed system. Therefore, a balance needs to be achieved between C and A.

✓ Consistency

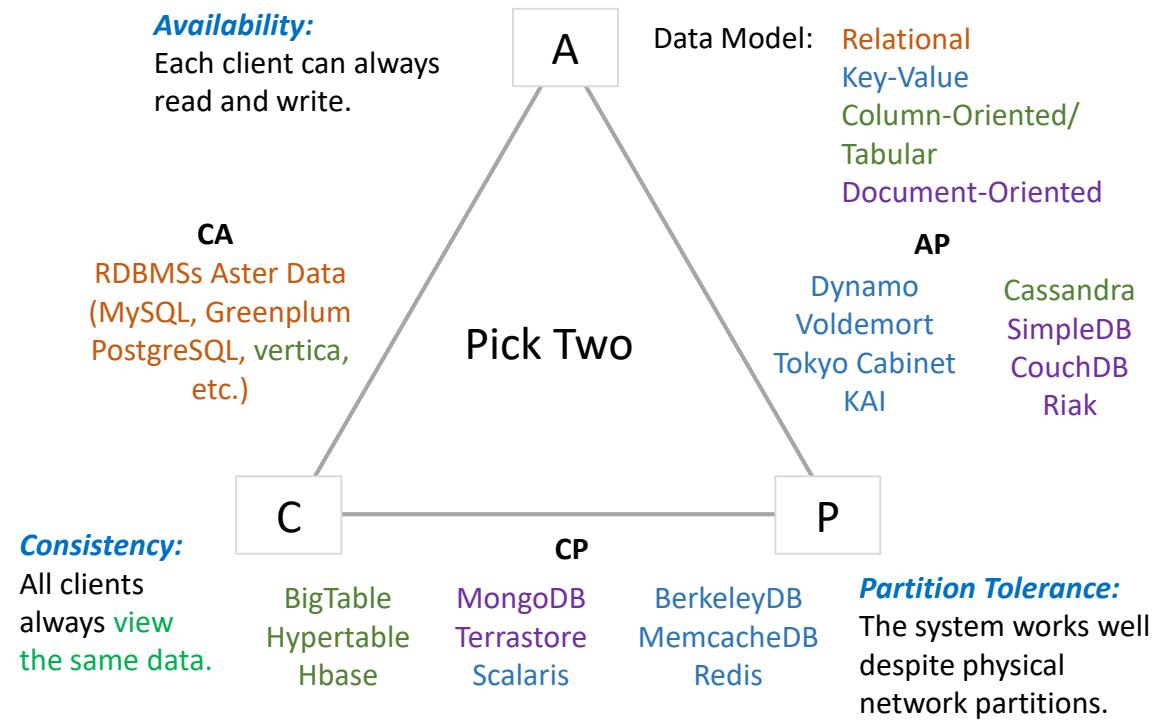
Consistency indicates that after the update operation is successful and the result is returned to the client, the data of all nodes at the same time is consistent. The notion of consistency is different from that in the ACID feature.

✓ Availability

It means that a timely and correct response can be made for each request, but it is not guaranteed that the request result is based on the latest written data.

✓ Partition tolerance

Partition fault tolerance means that a distributed system can still provide services that meet consistency and availability requirements when a node or network partition is faulty.

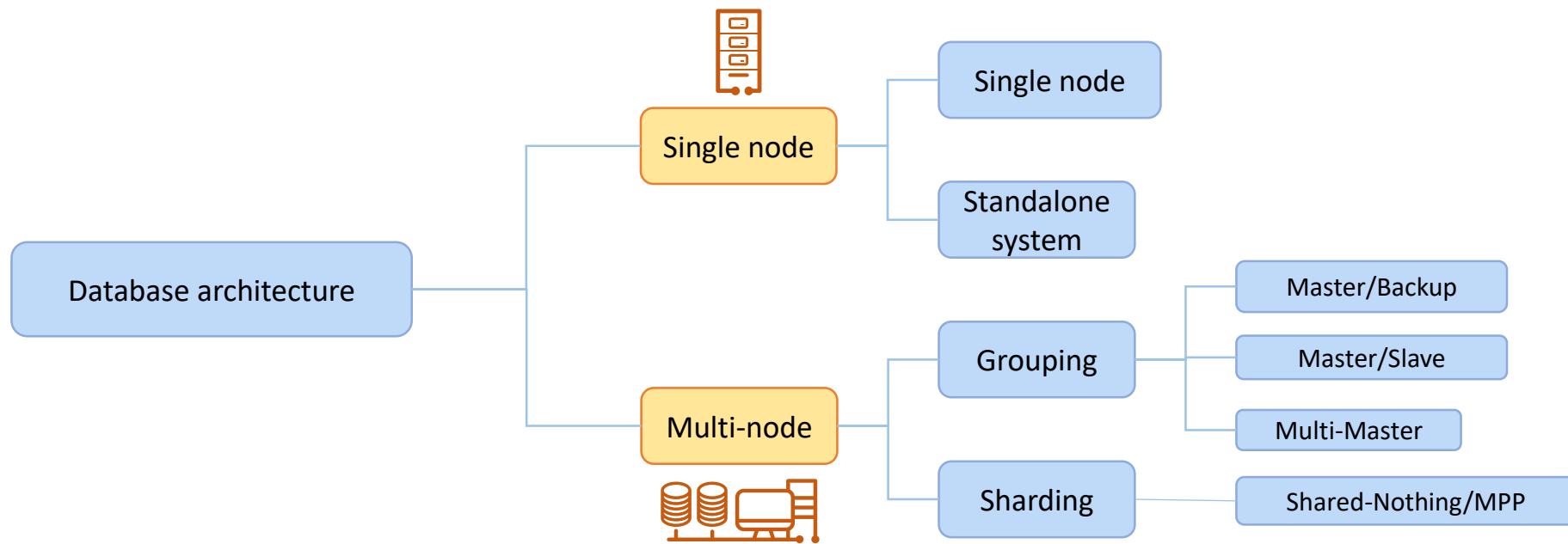


NewSQL

- NewSQL is defined as the development direction of next-generation databases, which refers to all various new scalable and high-performance databases. It has the massive storage management capability of the NoSQL database and the ACID feature and SQL convenience of the relational database.
- Simply speaking, SQL + NoSQL = NewSQL. Although the internal structure of NewSQL systems varies greatly, all the NewSQL systems have three significant common features. They all support the relational data model, use SQL as the main interface, and meet the features of the distributed database.
- The internal structure of NewSQL systems varies in supporting the relational database transaction feature and the distributed database feature. You can learn it from the architecture, SQL engine, and sharding mode aspects.
 - **New architecture:** Representative databases are Google Spanner, VoltDB, Clustrix and NuoDB which work on distributed nodes in a cluster, data is stored in shards, and SQL query is performed on shards on different nodes.
 - **SQL engine:** Representative databases are TokuDB and MemSQL. This type of databases has a highly optimized SQL engine.
 - **Transparent sharding:** Representative databases are ScaleBase, dbShards, and Scalearc. The system provides a sharding middleware layer. Data is automatically split and executed on multiple nodes.

Relational Database Architecture

As the service scale increases, the amount of data stored in the database and the service pressure on the database increase. Therefore, the database architecture needs to be changed to provide stable and efficient data services for upper-layer applications.

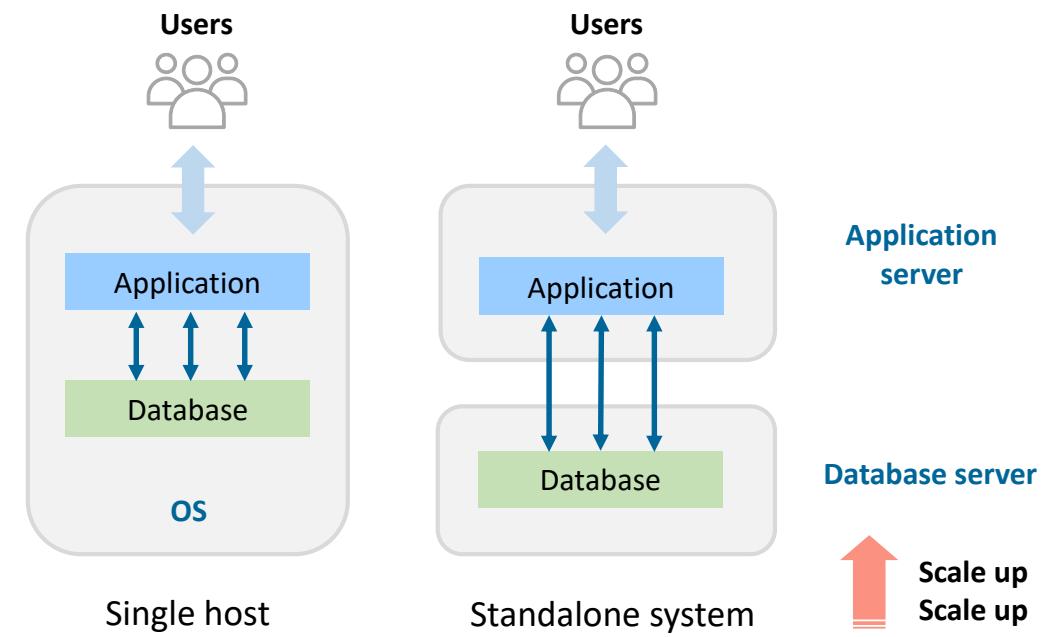


Evolution 1: Separation of database read and write; ***evolution 2:*** Database vertical sharding; ***evolution 3:*** Database and table horizontal sharding.

Single Node

To prevent application services and database services from competing for resources, the single-node database architecture evolves from the single-host mode to the standalone mode, separating applications from data services. For application services, you can increase the number of servers, balance the load, and enhance the concurrent capability of the system.

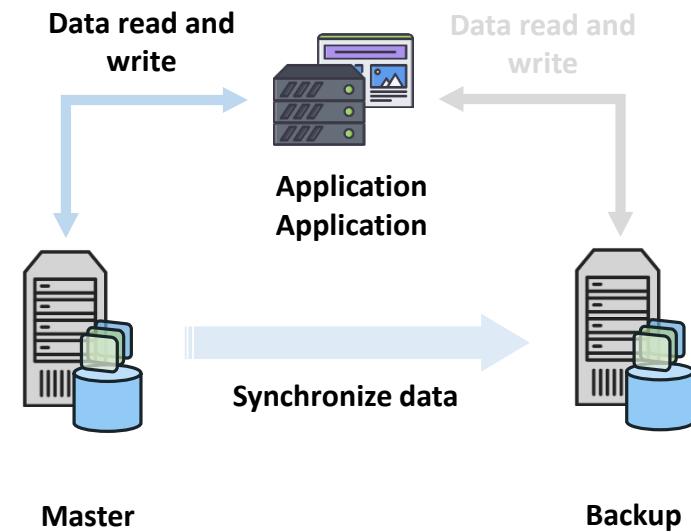
- ***Advantage:*** centralized deployment and easy O&M.
- ***Disadvantages:***
 - **Scalability**: The single-node database architecture can only be scaled up. That is, the performance can only be improved by adding hardware configurations. However, the hardware resources that can be configured for a single host may reach the upper limit.
 - **Single point of failure (SPOF)**: During scale-up, services need to be stopped. If a hardware fault occurs, the entire service is unavailable and even data is lost.
 - **Single-node performance bottleneck**



Grouping - Master/Backup

Master/Backup architecture:

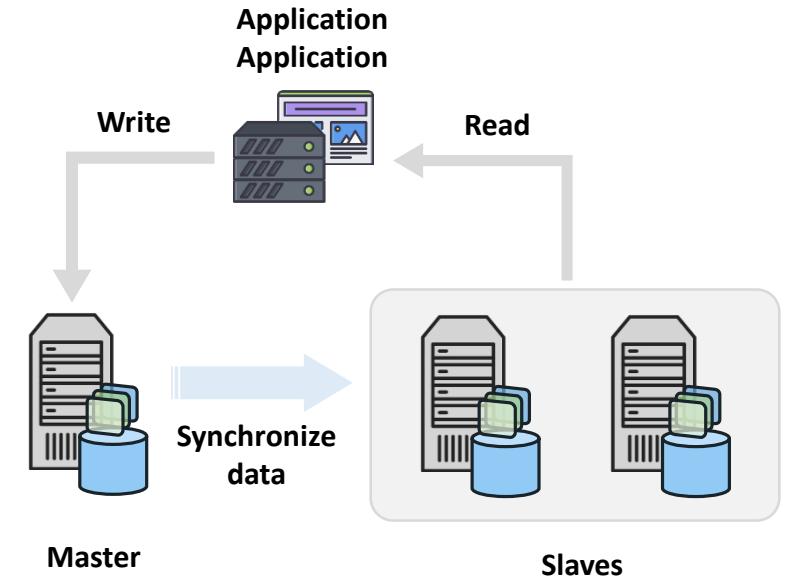
- The database is deployed on two servers. The server that provides data read and write services is called the master server. The other server uses the data synchronization mechanism to copy the data of the primary server, which is called a backup server. Only one server provides data services at a time.
- Advantages**
 - No extra effort needs to be paid during application development against database faults.
 - Compared with the single-node architecture, this architecture improves data error tolerance.
- Disadvantages**
 - Resources are wasted. The master and backup servers have the same configuration. However, resources in the standby server are basically idle for a long time and cannot be utilized.
 - The performance pressure is still concentrated on a single node, which cannot solve the performance bottleneck problem.
 - When a fault occurs, the switchover between the master and backup servers requires manual intervention or monitoring.



Grouping - Master/Slave

Master/Slave architecture

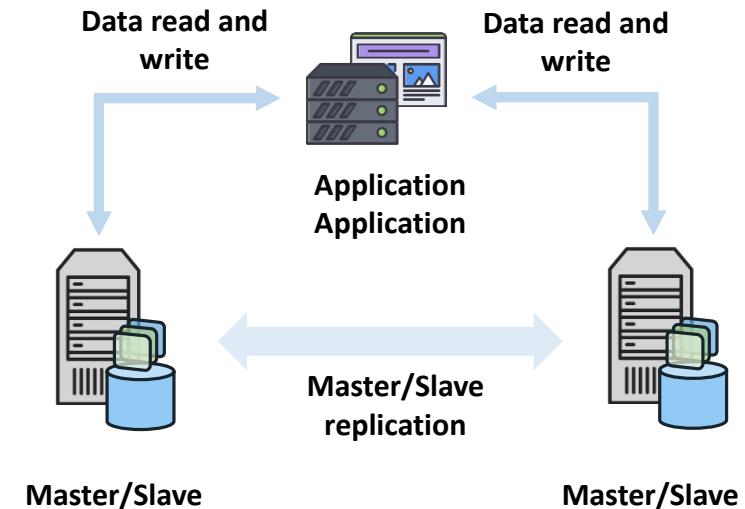
- The deployment mode is similar to the master/backup mode. However, the backup server becomes the slave server and provides certain data services. Pressures are balanced by read/write isolation:
 - Write, modify, and delete operations are performed on the write database (master server), and query requests are allocated to the read database (slave server).
- Advantages**
 - It improves resource utilization and applies to scenarios where read operations are more frequent than write operations.
 - In high-concurrency read scenarios, load balancing is employed to balance loads among servers.
 - The scalability of slave servers is flexible, and scaling does not affect services.
- Disadvantages**
 - There is a delay when data is synchronized to slave databases. As such, applications need to tolerate temporary inconsistency. This mode is not applicable to scenarios that have high consistency requirements.
 - The performance pressure of write operations is still concentrated on the master server.
 - If the master server is faulty, a master/slave switchover is required, where manual intervention requires response time and automatic switchover is complex.



Grouping - Multi-Master

Two database servers can function as the master and slave servers for each other.

- ***Advantages***
 - This improves resource utilization and reduces the risk of SPOF.
- ***Disadvantages***
 - Data is written to both servers and must be synchronized bidirectionally. Bidirectional replication also causes latency, and data may be lost in extreme cases.
 - With the increase of database servers, the data synchronization becomes very complex. In actual practice, the two-server cluster mode is commonly used.



Shared Disk

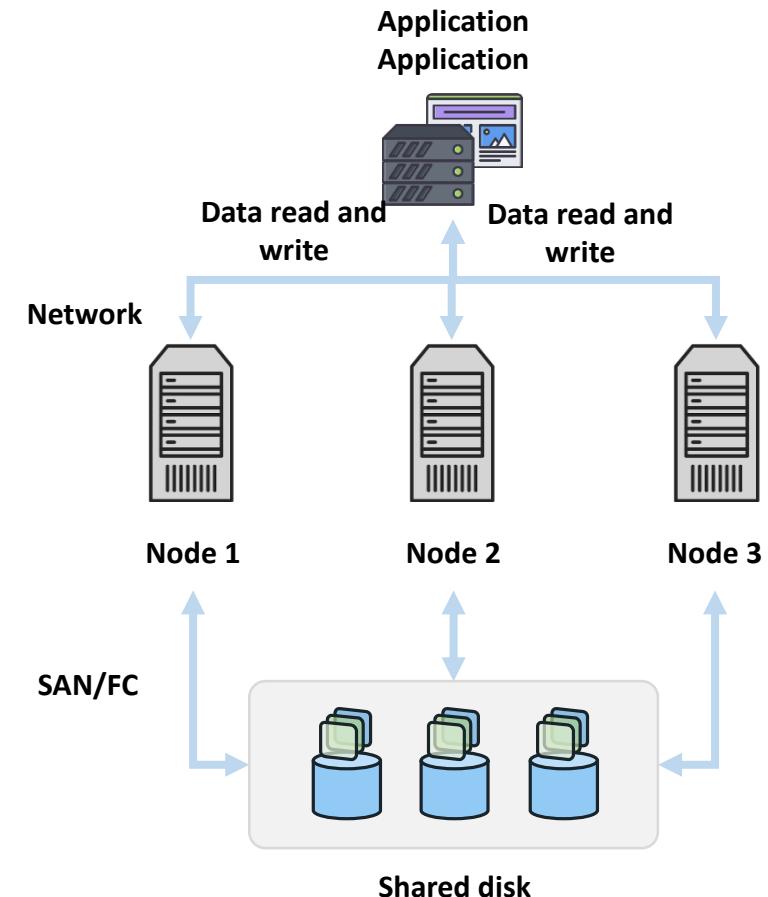
Shared disk is a special multi-master architecture. Database servers share data storage, and load balancing is implemented among multiple servers.

- ***Advantages***

- Multiple compute servers are deployed providing high availability. Scalability is achieved, avoiding the SPOF of the server cluster.
- Convenient scale-out can improve the parallel processing capability of the entire system.

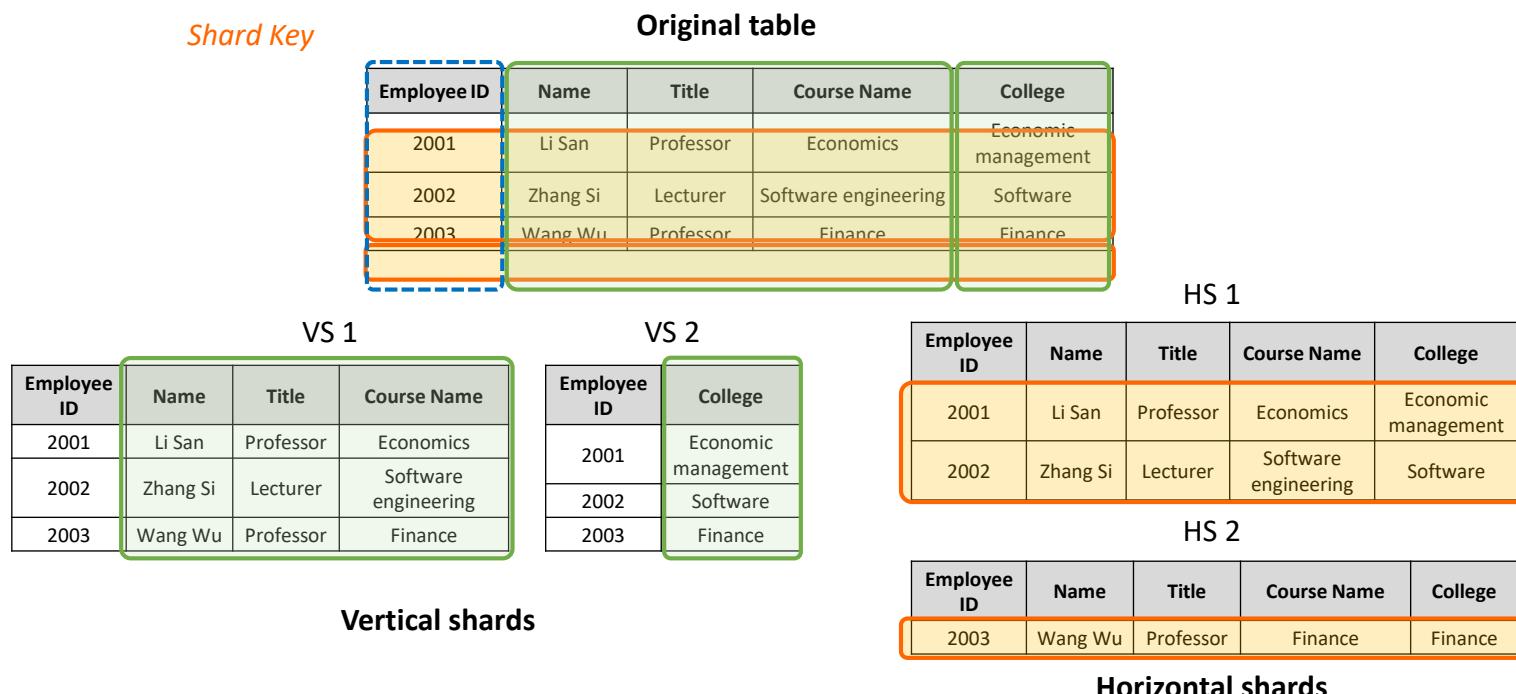
- ***Disadvantages***

- Difficult to implement.
- When the storage interface bandwidth is saturated, adding servers does not improve performance. The storage I/O may become the performance bottleneck of the entire system.



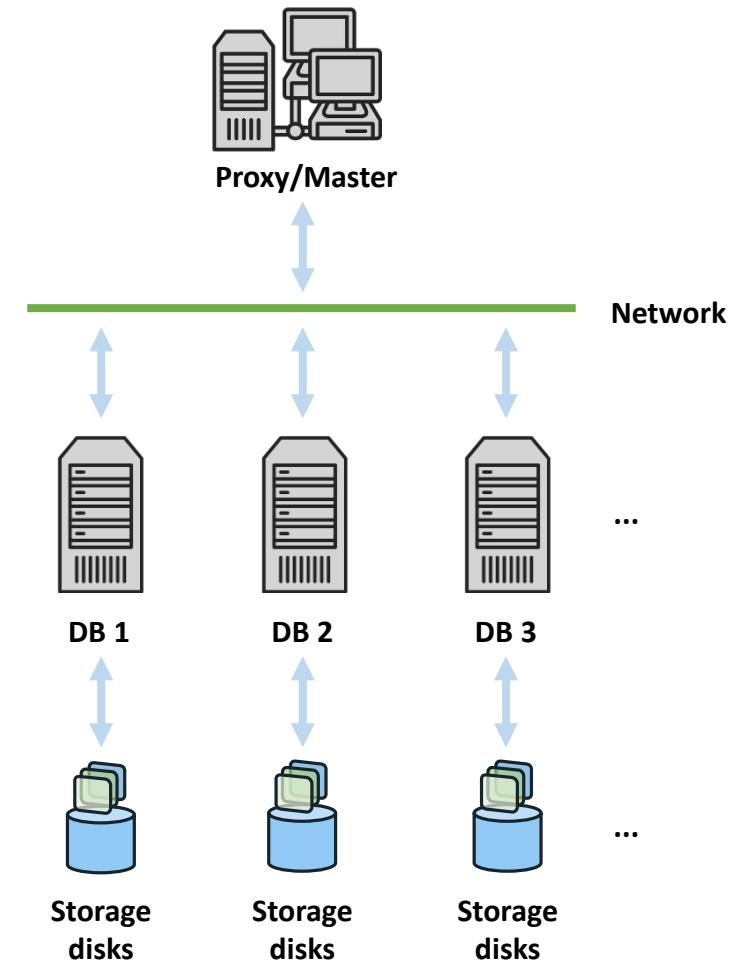
Sharding

- Sharding is a database architecture mode based on horizontal partitioning. In this mode, rows in a table are partitioned into different tables (called extents). Each extent has the same schema and columns, but each table has completely different rows. Similarly, the data stored in each extent is unique and independent of the data stored in other extents.
- It may be helpful to understand from the aspect of relationship between horizontal partitioning and vertical partitioning. In a vertical partitioning table, all columns are separated and put into a new, distinct table. Data in each vertical shard (VS) is independent of data in all other shards, and each shard contains different rows and columns.



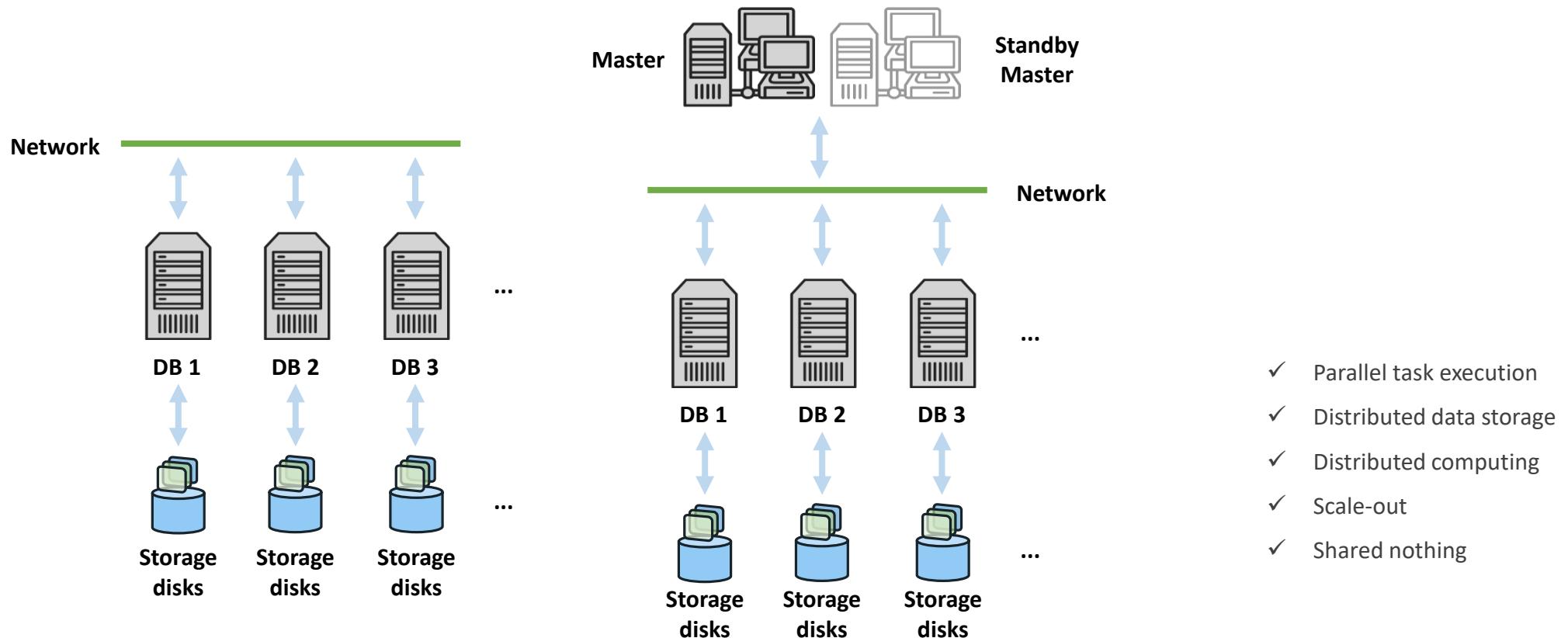
Shared Nothing

- Each processing unit has its own CPU, memory, and hard disk, and these units do not share resources, similar to the MPP mode. These processing units communicate with each other using protocols, achieving better parallel processing and scalability.
- Nodes are independent of each other and each node processes its own data. The processing results may be summarized and sent to the upper layer, or transferred among different nodes.
- The shared nothing system usually distributes its data to different databases on multiple nodes (different computers process queries from different users) or requires each node to retain its own application data backup by using some coordination protocols. This is usually called database sharding.



Massively Parallel Processing (MPP)

MPP distributes tasks to multiple servers and nodes in parallel. After the calculation on each node is complete, the results of each node are summarized to obtain the final result. It features high performance, high availability, and high scalability. It also provides a cost-effective universal computing platform for hyperscale data management and is widely used to support various data warehouse systems, BI systems, and decision support systems.



Typical Applications of Relational Databases



Online Transaction Processing (OLTP)

- OLTP is a typical application of traditional relational databases. It is a highly-available online system that processes basic and routine transactions, such as bank transactions, mainly with small-sized transactions and small-sized queries.
- The OLTP system emphasizes on the database memory efficiency. When evaluating an OLTP system, check the number of transactions executed per second and the number of Execute SQL statements. It emphasizes the command rates of memory metrics, bound variables, and concurrent operations.

Online Analytical Processing (OLAP)

- OLAP is a typical application of the data warehouse system. It supports complex analysis, focuses on decision support, and provides query results that are easy to understand. In the OLAP system, the quantity of the executed statements is not an evaluation standard, because if the execution time of a statement is long, the read data is huge.
- The OLAP system focuses on data analysis. The evaluation standards are the throughput (bandwidth) of the disk subsystem, SQL execution duration, disk I/O, and partitioning.

Comparison Between OLTP and OLAP

Item	OLTP	OLAP
Feature	Operation processing	Information processing
Orientation	Transaction	Analysis
User	Clerks, DBAs, and database professionals	Analysis and decision-making personnel
Purpose	Routine operations	Long-term information and decision-making support
Data	Current and latest	Maintained periodically
Summary	Original, highly detailed	Consolidated and integrated
Working unit	Short and simple transactions	Complex query
Access	Read/Write	Mostly read
Focus	Data input	Information output
Operation	Index or hash on the primary keyword	Large-scale scan
Number of access records	Dozens	Millions
DB scale	100 MB to GB	100 GB to TB
Advantages	High performance and availability	High flexibility and end user autonomy
Measurement	Transaction throughput	Query throughput and response time

Contents

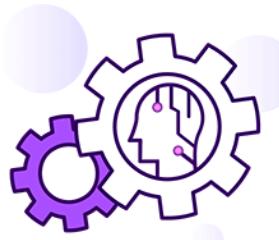
1. Database Overview
- 2. Introduction to openGauss**
3. openGauss Technical Specifications
4. Basic Functions

openGauss

As a leading database at enterprise level, openGauss is developed in collaboration with global partners. This multi-core-oriented open-source relational database provides ultimate performance, full-link service and data security, AI-based tuning, and efficient O&M capabilities. openGauss is released under the Mulan Permissive Software License v2. It deeply integrates Huawei's years of R&D experience in the database field and continuously builds competitive features based on enterprise-level scenario requirements.



High Reliability



Easy O&M



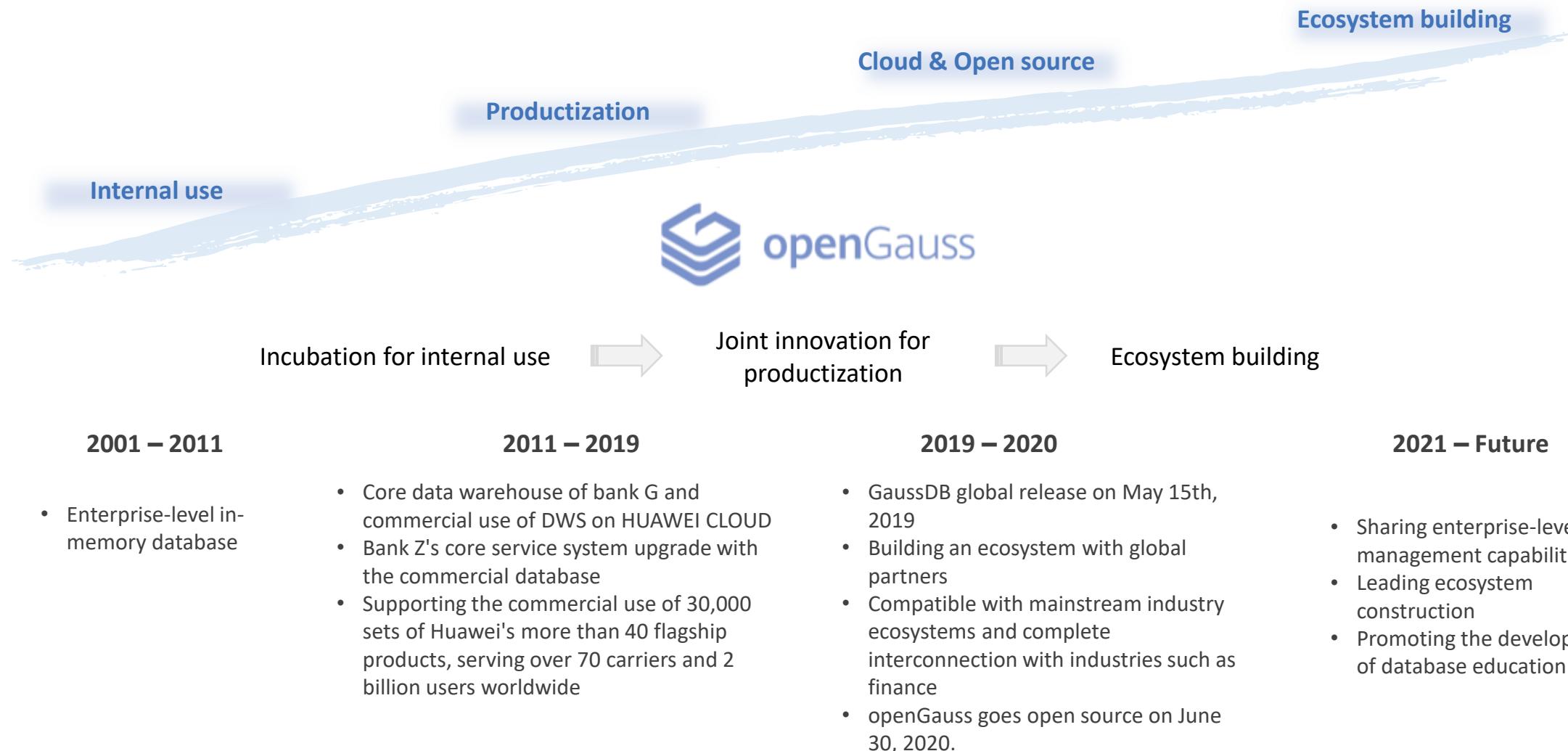
High Performance



High Security

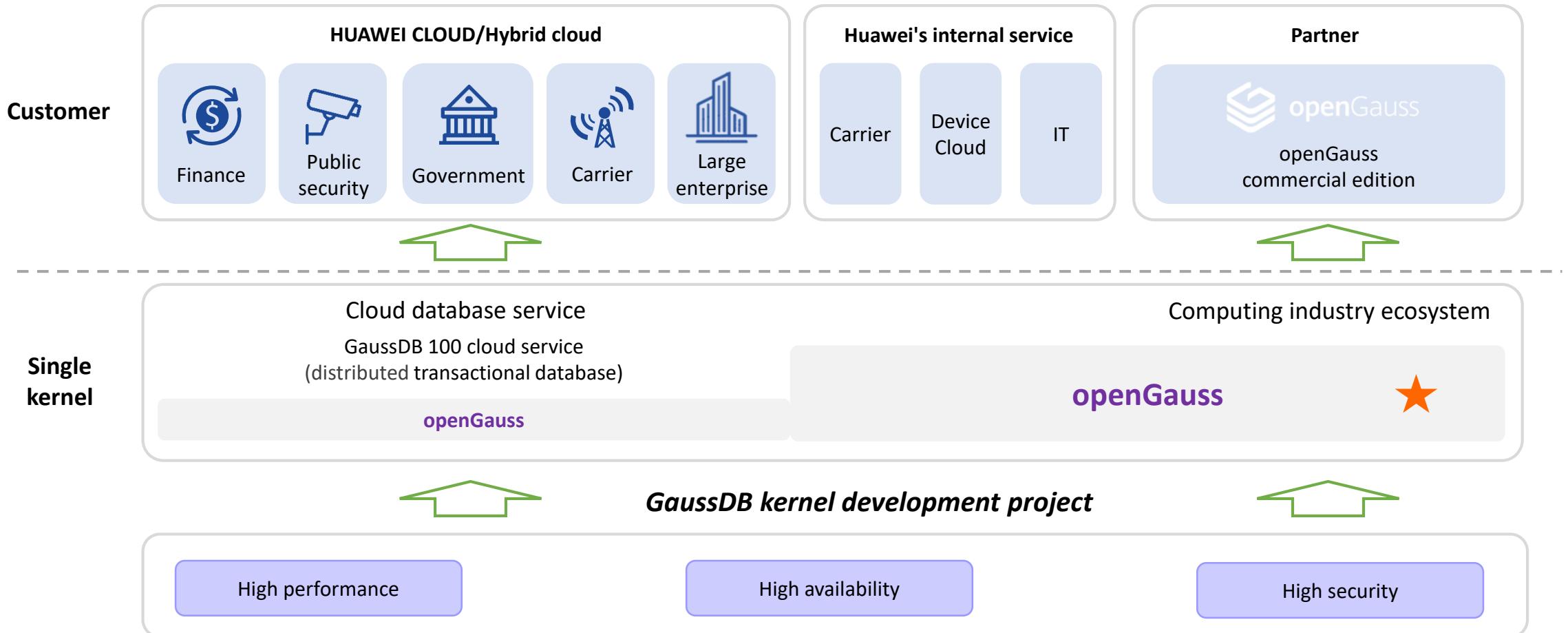


openGauss Evolution



openGauss: Commercial Use + Internal Use + Open Source, Long-Term Kernel Evolution

Huawei internal mappings, HUAWEI CLOUD GaussDB, and open-source openGauss share the code baseline.



openGauss Positioning

Users and Partners Access Enterprise-Class Database Capabilities

Value

openGauss provides multi-core ultimate performance, full-link service and data security, AI-based optimization, and efficient O&M capabilities. Huawei works with partners to build world-leading enterprise-class open-source relational databases.

Key features

High Performance

- ① Two-channel Kunpeng performance: 1,500,000 tpmC;
- Concurrency control technology for multi-core architecture;
- NUMA-Aware data structure
- SQL-Bypass intelligent traffic steering execution technology;
- Memory engine for real-time high-performance scenarios;
- ④

High Availability & Security

- ② Hitless service and RTO <10s;
- Refined security management: fine-grained access control and multi-dimensional audit;
- All-round data protection: encrypted storage, transmission, and export, dynamic masking, and fully encrypted computing;
- ⑤

Easy O&M

- ③ Optimizes parameters using AI and automatically recommends AI parameters.
- Provides slow SQL diagnosis and multi-dimensional self-monitoring to help you understand system performance in real time.
- Provides online self-learning SQL time prediction, quick locating, and quick optimization.

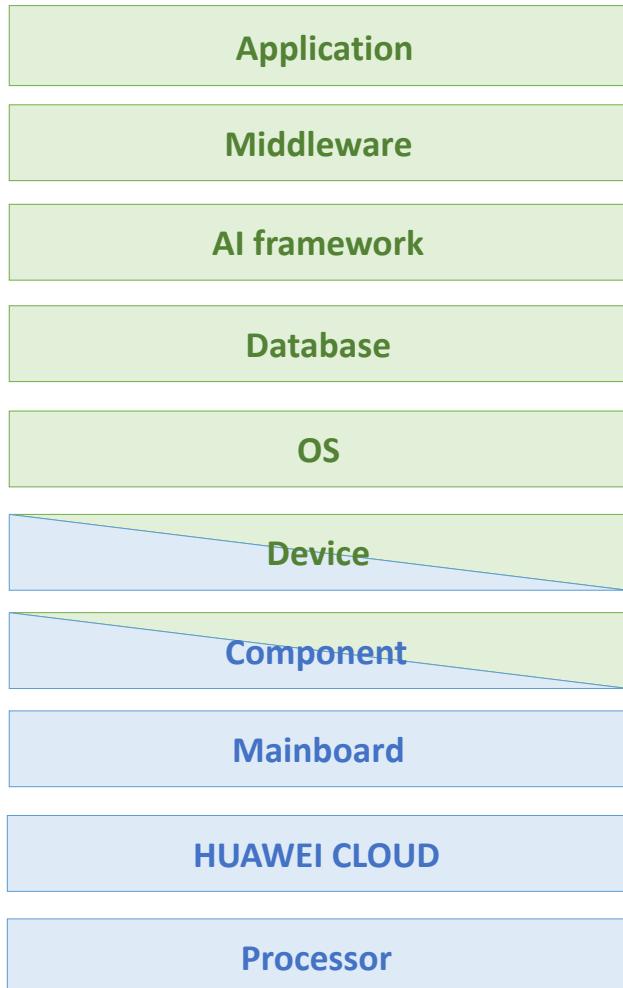
Full Openness

- Adopts the Mulan Permissive Software License, allowing code to be freely modified, used, and referenced.
- Opens database kernel capabilities.
- Open O&M monitoring, development, and migration tools.
- Open partner certification, training system, and university courses.

Huawei Computing Business Strategy

Partner

Huawei



Empowering partners

Enable 90% of software to run on Kunpeng in three years.

- Help partners migrate applications and software.

Open-source software

Redistribute software profits to new ISVs and redefine the software value chain.

- Open-source AI framework MindSpore
- openLooKeng open-source data virtualization engine
- openGauss open-source database (OLTP standalone edition), building a Kunpeng database ecosystem
- openEuler open-source OS, helping partners commercially release their OSs

Open hardware

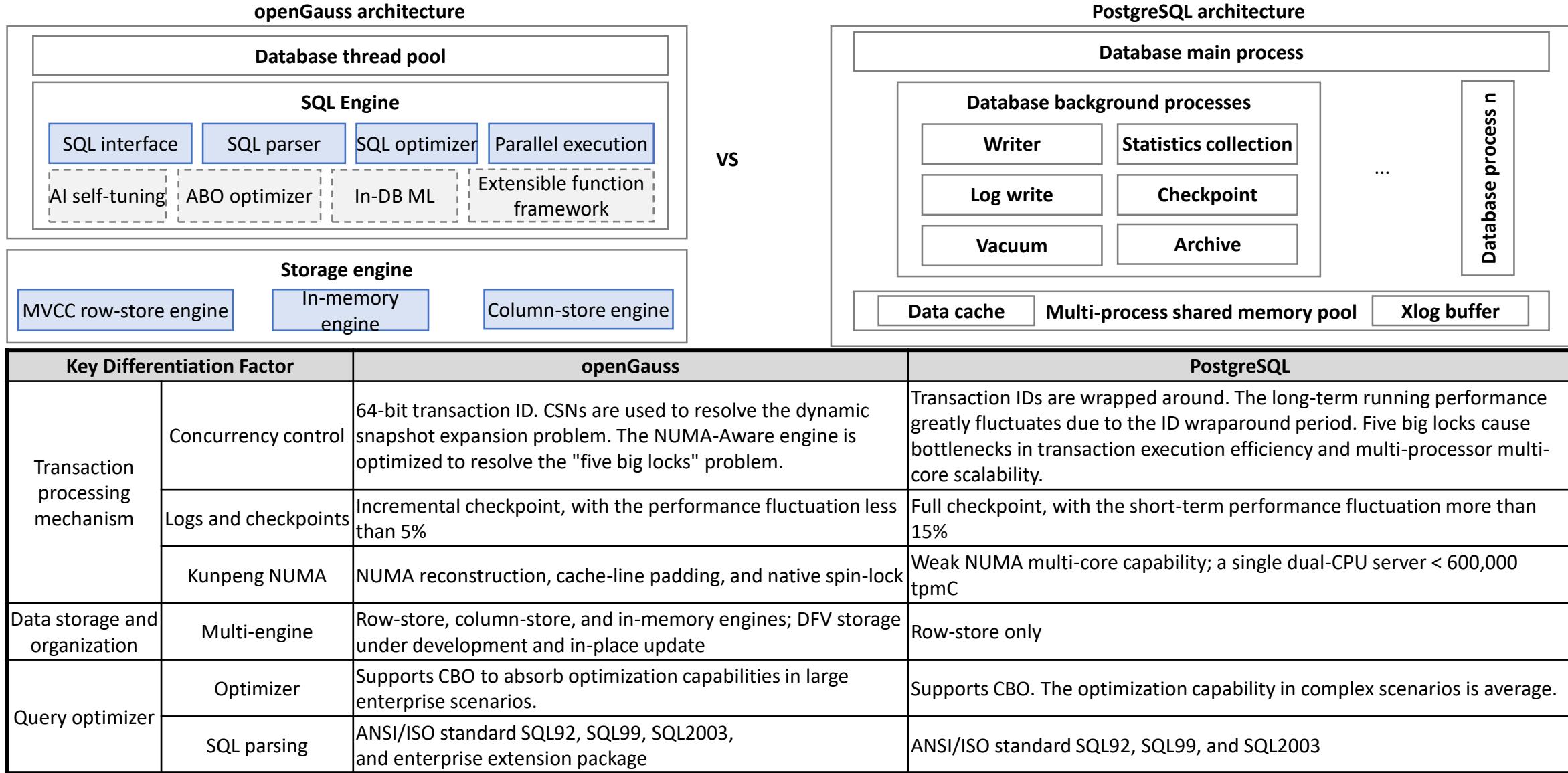
Lower the threshold, change market trends, and redefine the value chain of the entire system

- Provides open boards and components (such as SSDs, NICs, and RAID controller cards) based on Huawei Kunpeng processors for partners to develop their own components, servers, and PCs.

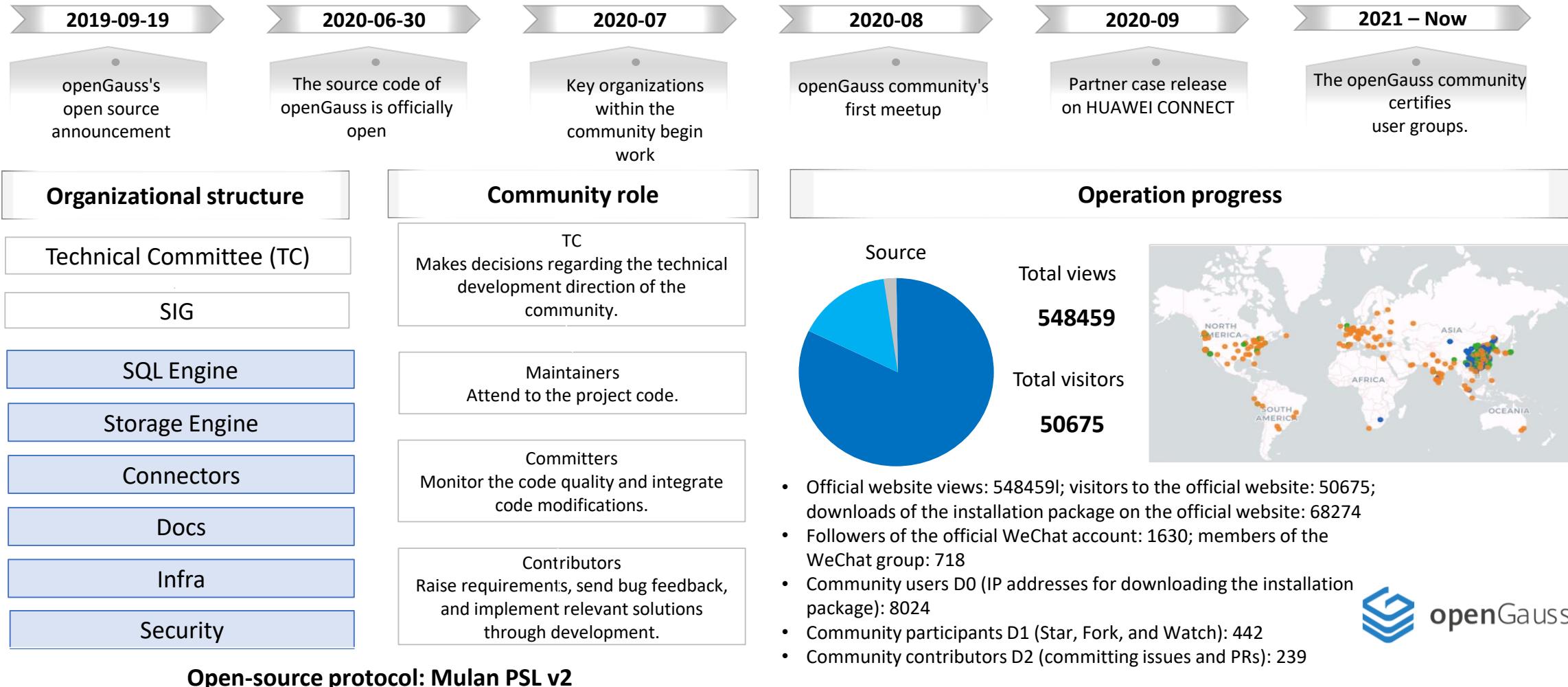
Huawei's focus

- Kunpeng processor R&D, all-scenario chips, and Kunpeng cloud services

Comparing openGauss and PG



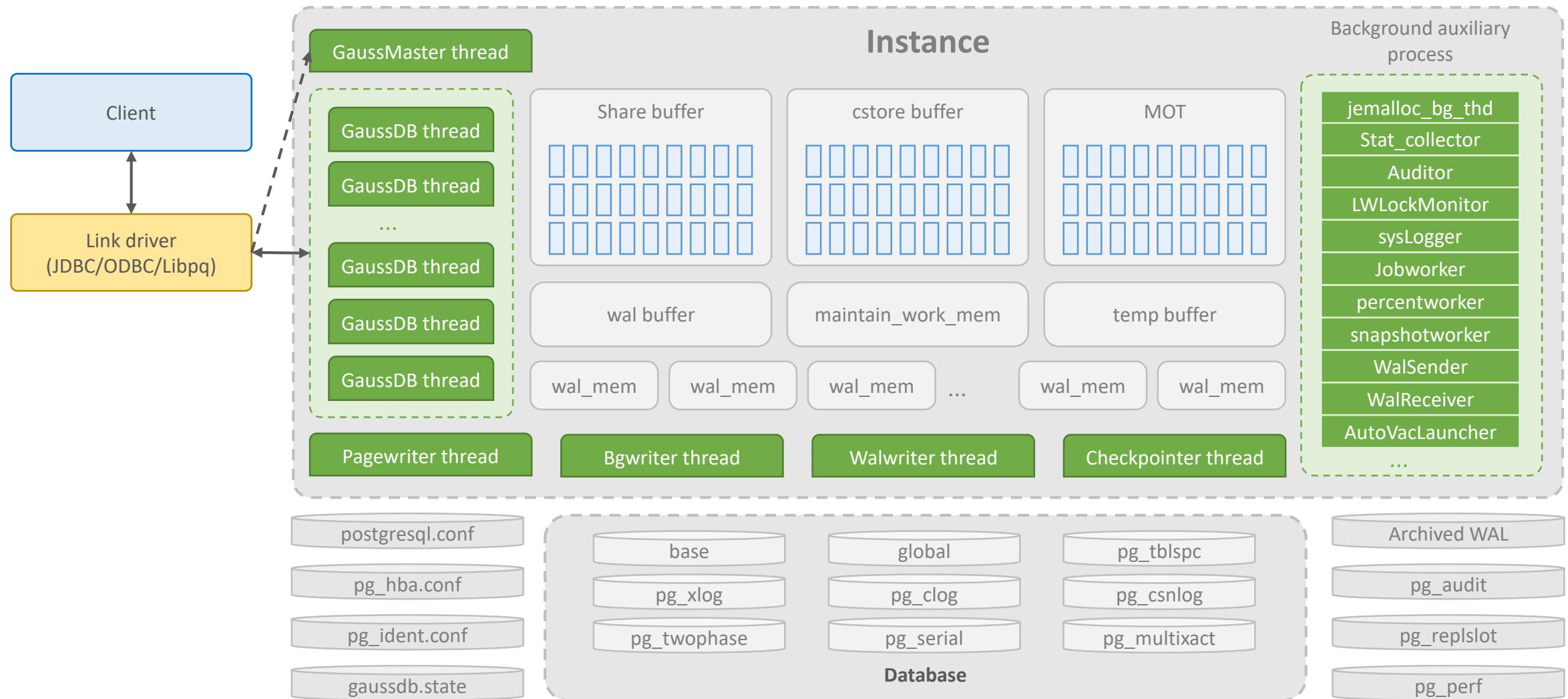
openGauss Community: Jointly Building an Ecosystem and Discussing the Development Direction



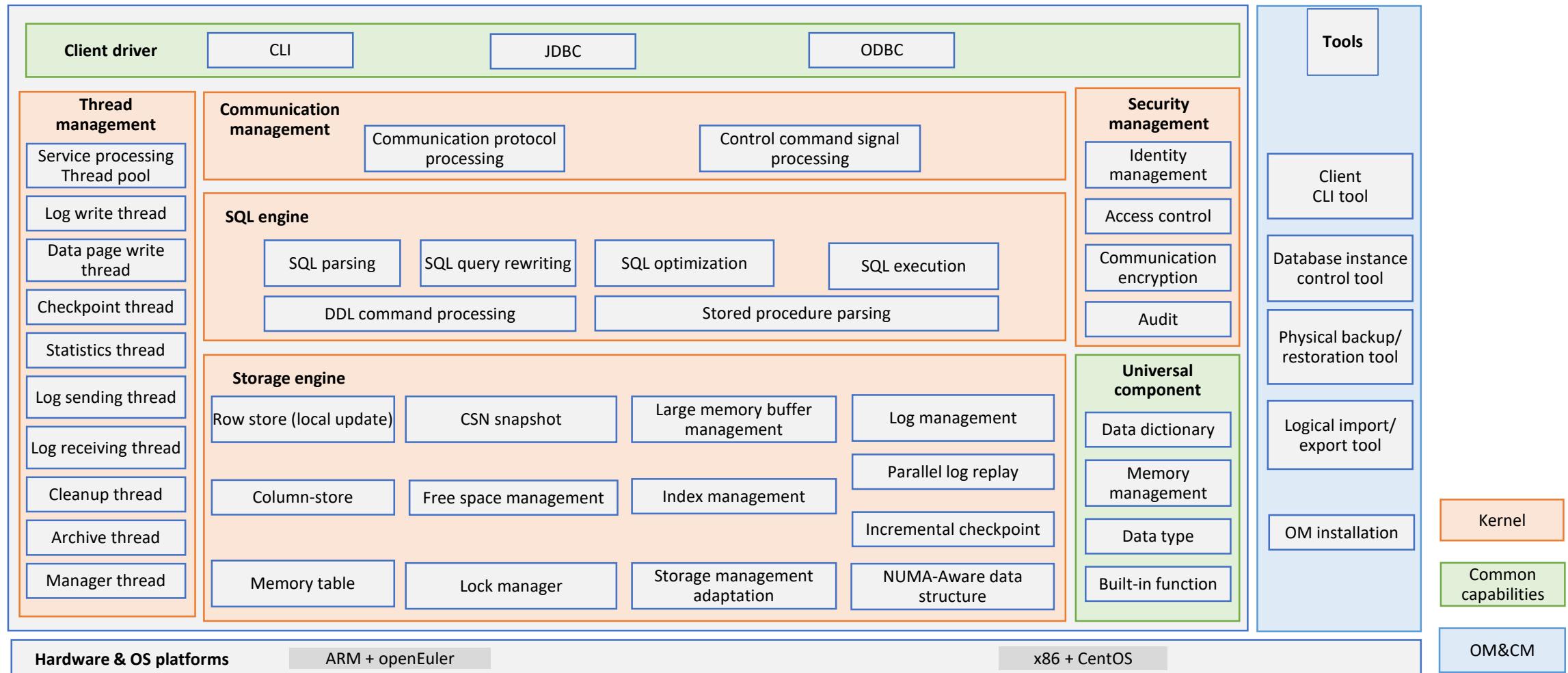
Contents

1. Database Overview
2. Introduction to openGauss
- 3. openGauss Technical Specifications**
4. Basic Functions

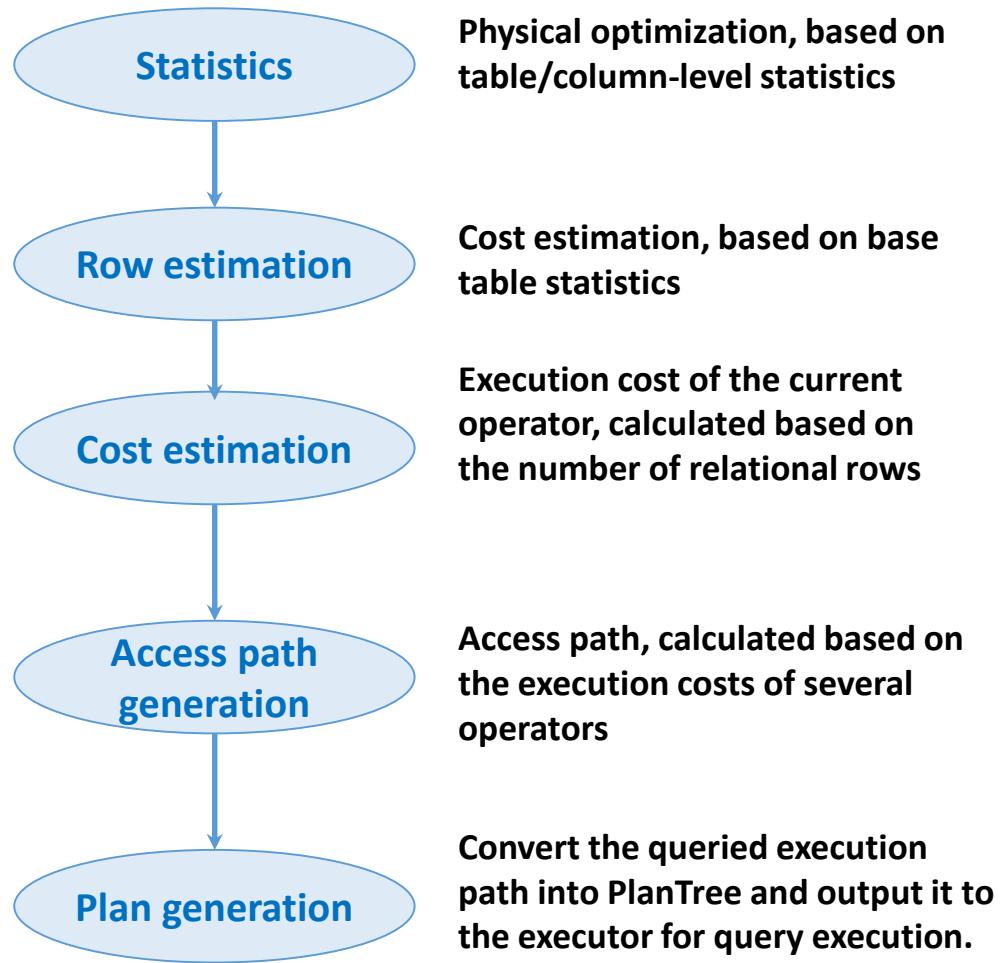
openGauss Architecture



openGauss Logic Modules



openGauss Query Optimization



Physical optimization technologies:

1. Table/Column-level statistics

Describes features of base table data, including unique values and MCV values, which are used to estimate the number of rows.

2. Row estimation

Estimates the size of the baserel, joinrel, and aggregation result sets to prepare for cost estimation.

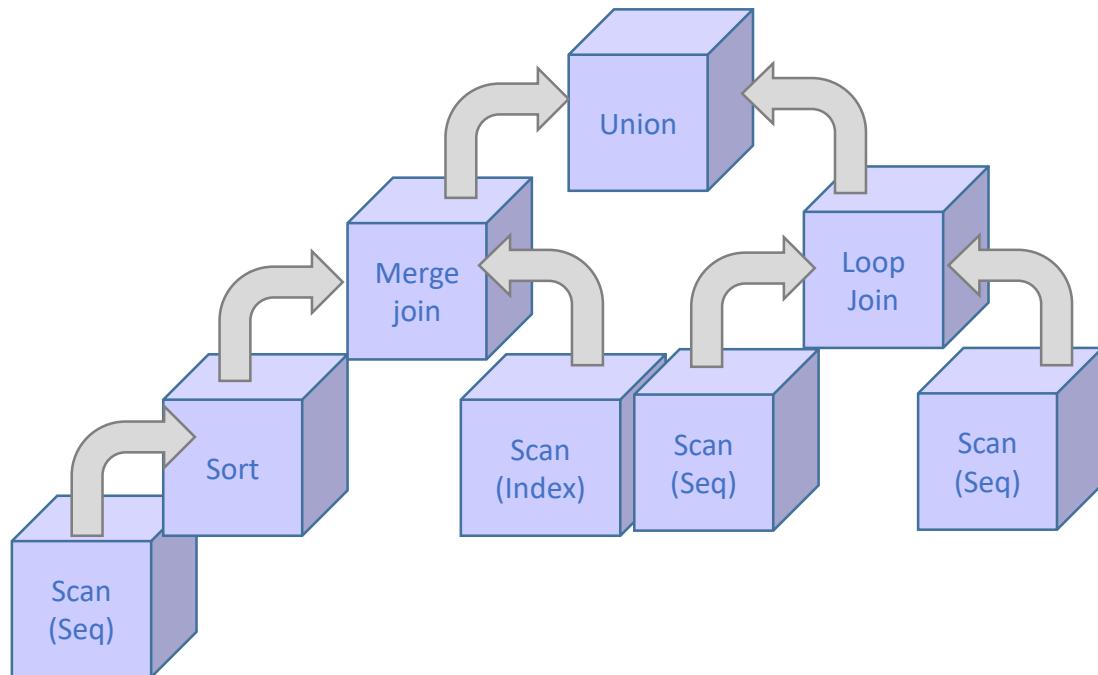
3. Cost estimation

Estimates the execution costs of different operators based on the data volume. The sum of the costs of all operators is the total plan cost.

4. Access path generation

Solves the optimal path algorithm (for example, dynamic programming and genetic algorithm) to deal with the access path search process, with the minimum search space to find the optimal access path.

openGauss Execution Engine



The relational database is used to calculate relation sets, and the execution engine is used as the control logic of the calculation and is implemented based on the relational calculation. The operators can be classified as follows:

1. **Scan plan node**

The scan node extracts data from the underlying data source, which may come from the file system or the network. Generally, the scan node is located on the leaf node of the execution tree and functions as the input source of execution data. Typical scan nodes include SeqScan, IndexScan, and SubQueryScan.

Key features: input data, leaf node, and expression filtering

2. **Control plan node**

Generally, control operators do not map algebra operators, and are introduced by executors to complete some special processes, such as Limit, RecursiveUnion, and Union.

Key feature: used to control the data process

3. **Materialize plan node**

These operators generally refer to algorithm requirements. When operator logic processing is performed, lower-layer data needs to be cached. As the amount of data returned by lower-layer operators cannot be predicted in advance, it is necessary to consider scenarios where data cannot be completely stored in the memory, such as Agg and Sort.

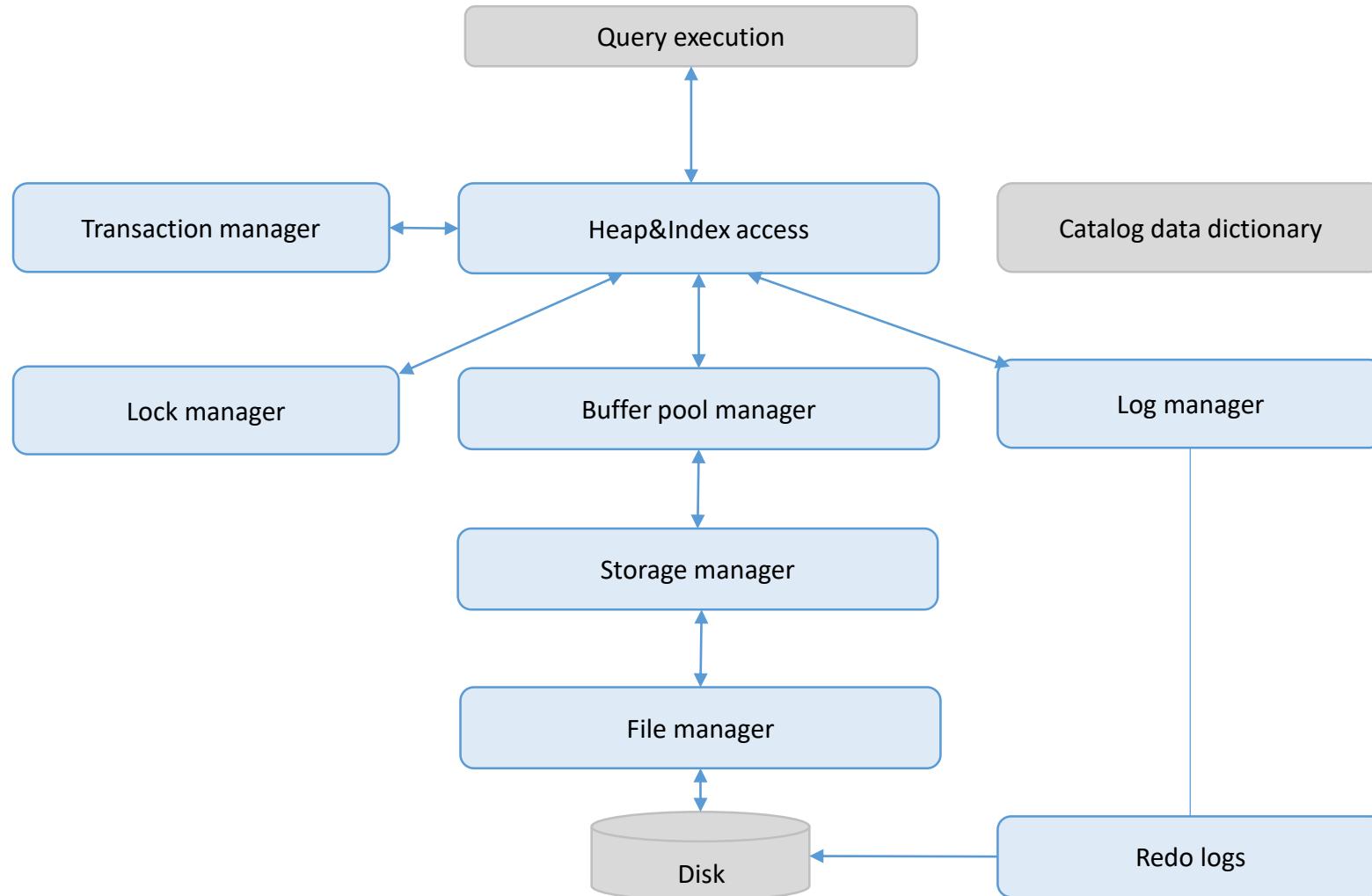
Key feature: Data is returned only after all data is scanned.

4. **Join plan node**

These operators are used to deal with the most common join operations in the database. They are classified into MergeJoin, SortJoin, and HashJoin based on the processing algorithm and data source.

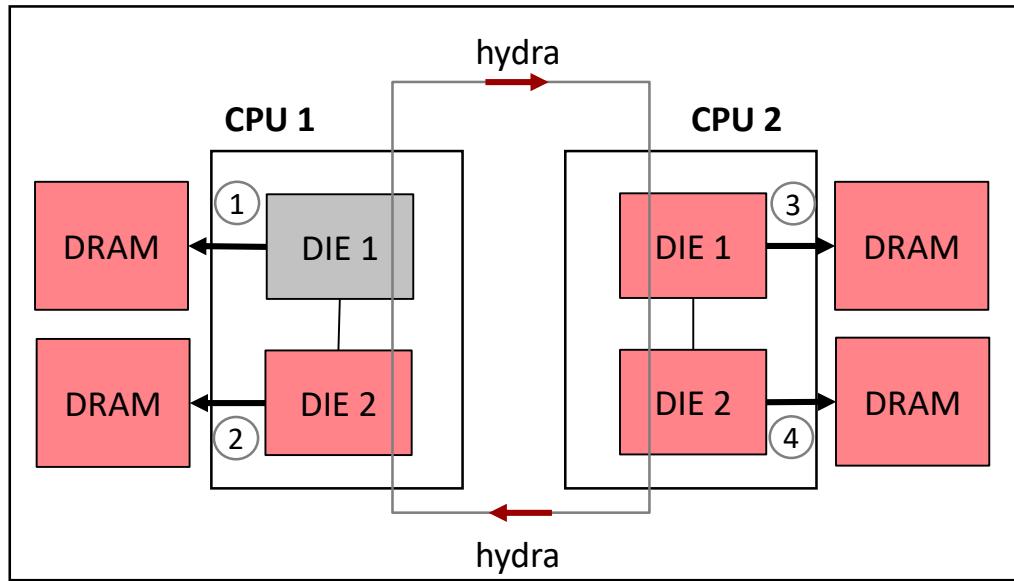
Key feature: multiple inputs

openGauss Storage Engine

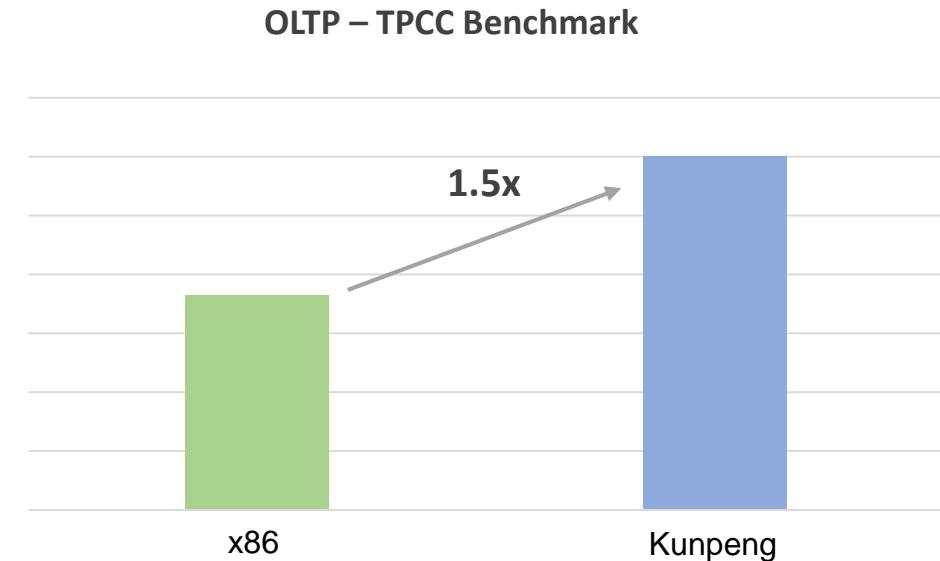


Optimal Performance of openGauss on Multi-core Servers

Typical Kunpeng multi-core CPU architecture

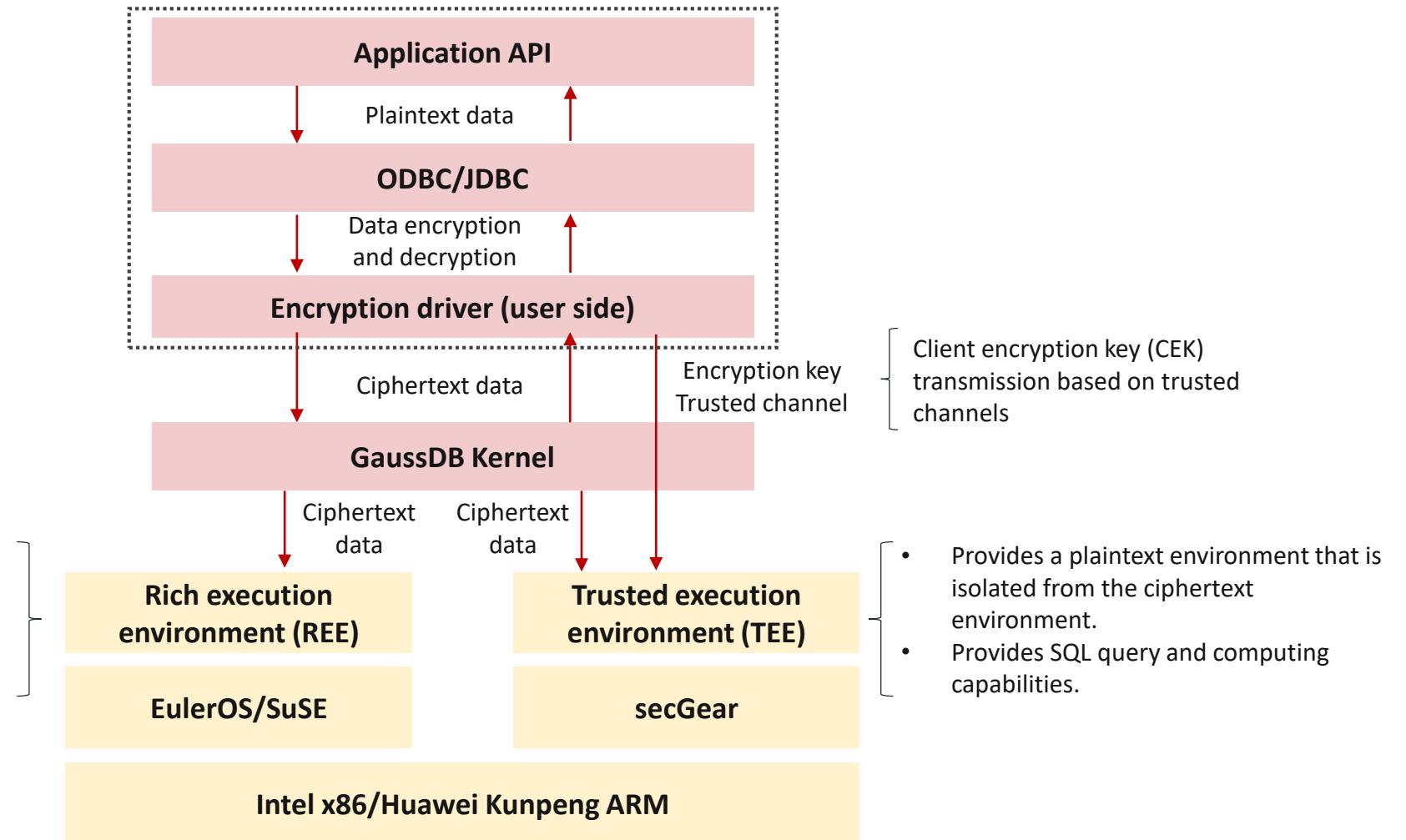


Two Kunpeng chips with 128 cores: 1,500,000 tpmC



Note: When two x86 Intel Xeon 6148 processors are used in the lab environment (configurations are identical except for the processors), the performance of various databases is as follows: Oracle < 1 million tpmC; PG < 700,000 tpmC; MySQL < 500,000 tpmC

openGauss Fully-encrypted Equality Query



openGauss Technical Specifications

Technical Specifications	Maximum Value
Database capacity	Varies depending on the OS and hardware
Size of a single table	32 TB
Size of data in a single row	1 GB
Size of a single field in each record	1 GB
Number of records in each table	2^{48}
Number of columns in a single table	250 to 1600 (varies with field type)
Number of indexes on a single table	Unlimited
Number of columns contained in a composite index	32
Number of constraints in a single table	Unlimited
Number of concurrent connections	10000
Number of partitions in a partitioned table	32768 (range partitioning)/64 (hash or list partitioning)
Size of each partition in a partitioned table	32 TB
Number of records in a single partition of a partitioned table	2^{55}

Contents

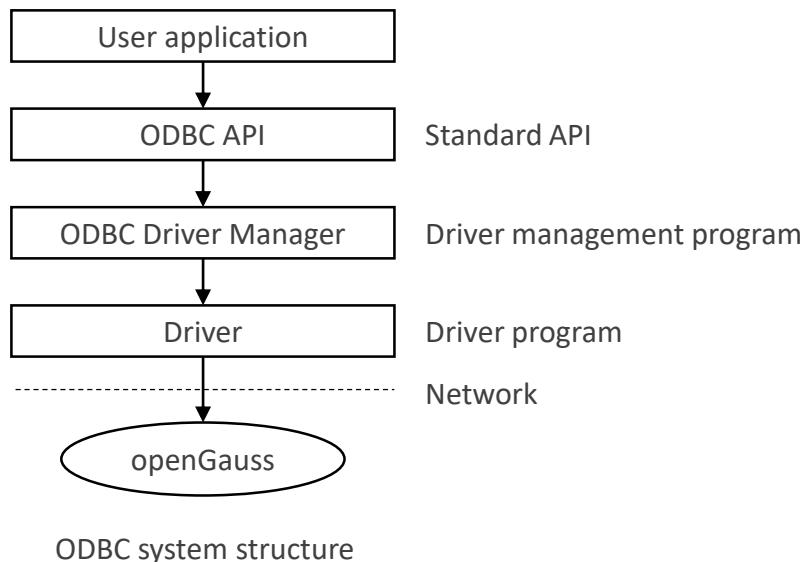
1. Database Overview
2. Introduction to openGauss
3. Technical Specifications
- 4. Basic Functions**

openGauss Supports Standard SQL

- SQL is a standard programming language used to control the access to databases and manage data in databases. SQL:2011 standards are classified into core features and optional features. Most databases do not fully support SQL standards.
- openGauss is a high-performance HA rational database that supports the SQL2003 standard and primary/standby deployment. In addition, openGauss supports most of the core features of SQL:2011 as well as some optional features, providing a unified SQL interface for users.
- With standard SQL, all database vendors use unified SQL interfaces, reducing the costs associated with learning languages and migrating applications.

openGauss Supports ODBC-based Development APIs

- Open Database Connectivity (ODBC) is a Microsoft API used to access databases based on the X/OPEN CLI. Applications interact with the database through the APIs provided by ODBC, which enhances their portability, scalability, and maintainability.
 - openGauss supports ODBC 3.5 in the following environments.
 - The ODBC Driver Manager running on UNIX or Linux can be either unixODBC or iODBC. Select unixODBC-2.3.0 here as the component for connecting to the database.
 - Windows has a native ODBC Driver Manager. You can locate **Data Sources (ODBC)** by choosing **Control Panel > Administrative Tools**.



OS	Platform
CentOS 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4	x86_64
CentOS 7.6	ARM64
EulerOS 2.0 SP2/SP3	x86_64
EulerOS 2.0 SP8	ARM64

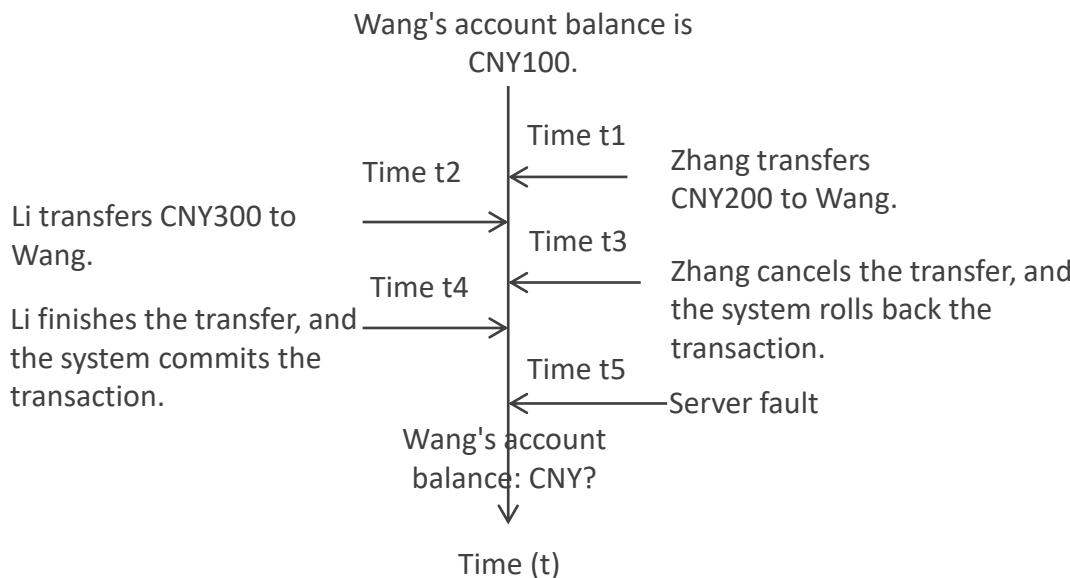
OS Supported by ODBC

openGauss Supports JDBC-based Development APIs

- Java Database Connectivity (JDBC) is a Java API used to run SQL statements. It provides unified access APIs for different relational databases, based on which applications process data. The openGauss library supports JDBC 4.0 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridging.
 - openGauss supports the JDBC 4.0 standard API.
 - Standard ODBC and JDBC APIs are provided to ensure quick migration of user services to openGauss.
 - A function to connect JDBC to a third-party log framework has been added, enabling JDBC to interconnect with a third-party log framework to meet the log management and control requirements of users.

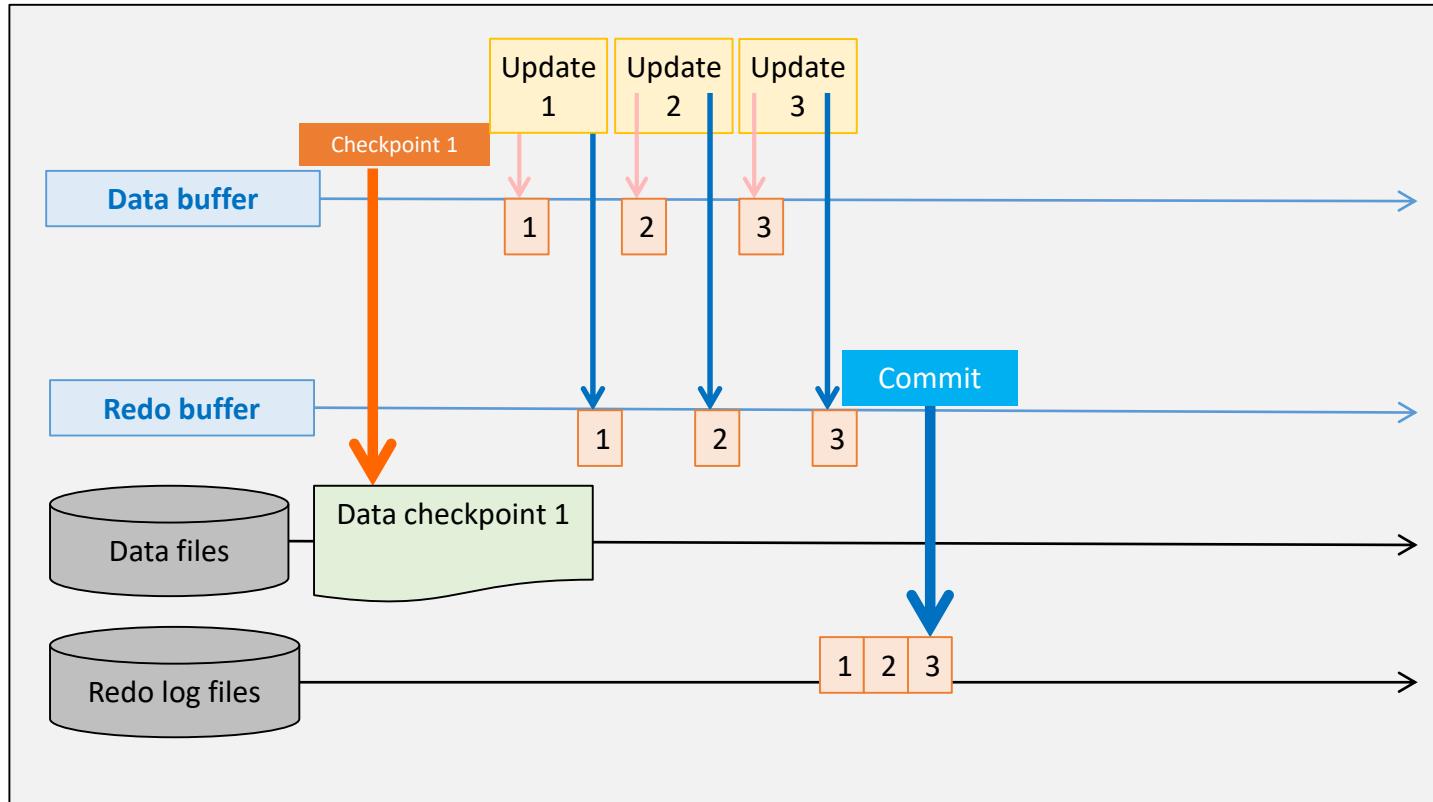
openGauss Database Transaction

- Transaction **ACID**
 - *Atomicity (A)*: ensures that each transaction is treated as a single "unit", which either succeeds completely, or fails completely.
 - *Consistency (C)*: ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants.
 - *Isolation (I)*: ensures that an unfinished transaction does not expose its result to subsequent other transactions before it is committed.
 - *Durability (D)*: ensures that the result of a committed transaction will not be lost due to faults which occur later.
- Example: Bank transfer (Zhang and Li transfer money to Wang.)



- The ACID features ensure that Wang's account balance is CNY400 after the server restarts.
- Atomicity: After Zhang cancels the transfer, the account balance of Wang is still CNY100.
- Consistency: Li's account balance is CNY300 less, and Wang's is CNY300 more. The sum of the two accounts remains unchanged.
- Isolation: The CNY200 transferred by Zhang is invisible to Li. Otherwise, the result is CNY700.
- Durability: After the server is restored from a fault, Li's transfer is still valid.
- Wang's account balance is CNY400.

openGauss Database Transaction Mechanism



1. Transaction durability is implemented by using the write-ahead log (WAL) algorithm. When a transaction is committed, a redo log is written to disks using the WAL method.
2. Checkpoint: Write the data in the dirty buffer queue to the data file.

openGauss Supports Functions and Stored procedures

- 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. openGauss supports functions and stored procedures compliant with the SQL standard. These stored procedures are compatible with certain mainstream stored procedure syntax, improving their usability and offering the following advantages:
 - Allows customers to modularize program design and encapsulate SQL statement sets, making them easy to invoke.
 - Caches the compilation results of stored procedures to accelerate SQL statement set execution.
 - Allows system administrators to restrict permission to execute a specific stored procedure and control access to the corresponding type of data. This prevents access from unauthorized users and ensures data security.

openGauss Is Compatible with PG APIs

- The basic PostgreSQL publication contains only two client APIs:
 - libpq is included because it is a C language API on which many other client APIs depend.
 - ECPG is included because it relies on the server-side SQL syntax and is therefore very sensitive to changes in PostgreSQL.
- openGauss is compatible with the PostgreSQL clients and standard APIs, and can be seamlessly interconnected with PostgreSQL ecosystem tools.

openGauss Supports SQL Hints

- Plan hints enable you to specify a join order. You can also join, stream, and scan operations, the number of rows in a result, and redistribution skew information to tune an execution plan, improving query performance.
- openGauss supports SQL hints, which can override any execution plan and thus improve SQL query performance.

Quiz

1. (Single-choice) Which of the following is not a component of a database system?
 - A. Database:
 - B. DBMS
 - C. Database application
 - D. Database storage medium
2. (Single-choice) Which of the following statements about the ACID feature of database transactions is incorrect?
 - A. A indicates atomicity, whereby operations in a transaction are either all successful or all failed.
 - B. C refers to consistency, whereby the system status can only be the status before the transaction or after the transaction is successful. No inconsistent intermediate status can occur.
 - C. I refers to availability, whereby the database system must provide the highest availability possible for transaction execution in order to ensure as many successfully executed transactions as possible.
 - D. D refers to durability, whereby the status of a successful transaction can be maintained, even if the machine is powered off.

Quiz

3. (Multiple-choice) Which of the following phases are involved in the development of data management technologies?
- A. Manual management
 - B. File system
 - C. DBS
 - D. AI management

Summary

- This chapter describes database development and evolution, database classification and data models, database system composition, and the openGauss database. This document describes the database definition and technology development history, architecture evolution of relational databases, main application scenarios, development history of openGauss database, product positioning, basic indicators, and basic functions.

Recommendations

- HUAWEI CLOUD:
 - <https://www.huaweicloud.com/>
- openGauss help document:
 - <https://opengauss.org/zh/docs/2.0.0/docs/installation/installation.html>
- openGauss code repository:
 - <https://gitee.com/opengauss>

Acronyms and Abbreviations

Acronym and Abbreviation	Full Spelling
DB	Database
DBMS	Database management system
DDL	Data definition language
DML	Data manipulation language
DBA	Database administrator
DBS	Database system
SQL	Structured query language
MPP	Massive parallel processing
OLTP	On-line transaction processing
OLAP	On-line analytical processing
DWS	Data warehouse service
RTO	Recovery time objective
JDBC	Java Database Connectivity
ODBC	Open Database Connectivity
WAL	Write-ahead log
CLI	Command-line interface
tpmC	transactions per minute C

02 Database Installation and Deployment



Foreword

openGauss is a relational database. It uses a client/server, single-process multi-thread architecture and supports standalone and one-primary multi-standby deployment. When primary/standby deployment is used, the standby node can be read too, and HA and read expansion are supported. This chapter describes how to set up the environment, how to install and deploy a database, and the functions and applications of openGauss-related O&M tools.

Objectives

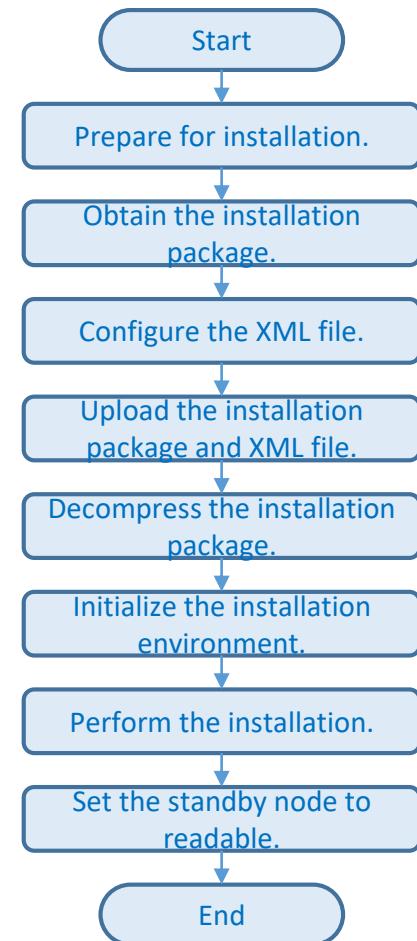
- Upon completion of this course, you will:
 - Understand the installation and deployment of the openGauss database.
 - Understand openGauss database connection and authentication.
 - Understand the functions and operations of common openGauss database tools.
 - Understand how to uninstall an openGauss database.

Contents

- 1. openGauss Installation**
2. Connection and Authentication
3. openGauss Tools
4. Database Uninstallation

openGauss Installation Process

Process	Description
Prepare for installation.	Before the openGauss installation, prepare the software and hardware environment and complete necessary configurations. This document provides the minimum requirements for the openGauss installation.
Obtain the installation package.	You need to download the installation package from the openGauss open-source community and check the package content.
Configure the XML file.	Before installing openGauss, create an XML configuration file. The configuration file contains information about the server where openGauss will be deployed, the installation path, IP address, and port. This file guides how to deploy the openGauss. You need to configure the XML configuration file based on deployment requirements.
Initialize the installation environment.	To initialize the installation environment, upload the installation package and the XML file, decompress the installation package, and use <code>gs_preinstall</code> to prepare the environment.
Perform the installation.	Use <code>gs_install</code> to install the openGauss.
Set the standby node to readable.	This operation is optional, but making the standby node readable improves data consistency.



Obtaining the Installation Package

- Step 1: Download the installation package of the corresponding platform from the openGauss open-source community.
 - Log in to the openGauss open-source community at <https://opengauss.org/zh/download.html>, select **2.0.0** in the **Version** field, and download the corresponding simplified installation package.
 - Click **Download**.
- Step 2: Check the installation package. Decompress the package and confirm that installation directory is correct and all the files are there.
 - Run the following command in the directory where the installation package is stored:

```
tar -jxf openGauss-x.x.x-openEuler-64bit.tar.bz2
ls -1b
```
 - The **ls** command should display information similar to the following:

```
total 90296
drwx----- 3 root root 4096 Mar 31 21:18 bin
drwx----- 3 root root 4096 Mar 31 21:18 etc
drwx----- 3 root root 4096 Mar 31 21:18 include
drwx----- 4 root root 4096 Mar 31 21:18 jre
drwx----- 5 root root 4096 Mar 31 21:18 lib
-rw----- 1 root root 92427499 Apr 1 09:43 openGauss-2.0.0-openEuler-64bit.tar.bz2
drwx----- 5 root root 4096 Mar 31 21:18 share
drwx----- 2 root root 4096 Mar 31 21:18 simpleInstall
-rw----- 1 root root 32 Mar 31 21:18 version.cfg
```

Hardware Requirements

- The following table lists the minimum hardware requirements for openGauss server. During hardware planning for the actual product, you have to consider how much data will be involved and the database response speed you want to see.

Item	Configuration
Memory	<p>It is recommended that there be at least 32 GB of memory available for function debugging.</p> <p>In performance testing and commercial deployment, it is recommended that there be at least 128 GB of memory for single-instance deployment.</p> <p>Complex queries require much more memory, so there may not be enough memory for high-concurrency scenarios. For high-concurrency scenarios, it is recommended that a large-memory server or load management be used to restrict concurrency on the system.</p>
CPU	<p>At least one 8-core 2.0 GHz CPU should be used for function debugging.</p> <p>For performance testing and commercial deployment, a single 16-core 2.0 GHz CPU is recommended for single-instance deployment.</p> <p>You can configure the CPUs for hyper-threading or non-hyper-threading.</p> <p>For individual developers, the minimum configuration is 2 cores with 4 GB of memory, but the recommended configuration is 4 cores and 8 GB of memory. Currently, openGauss supports only the Kunpeng-powered servers and x86_64-based universal PC servers.</p>
Disk	<p>Hard disks used for installing openGauss must meet the following requirements:</p> <ul style="list-style-type: none">At least 1 GB is used to install the openGauss applications.Each host requires at least 300 MB for data storage.More than 70% of the remaining space is used for data storage. <p>You are advised to configure the system disk to RAID 1 and data disk to RAID 5 and plan four groups of RAID 5 data disks for installing openGauss. RAID configurations are not described in this document. You can configure RAID by following instructions in the hardware vendor's manuals. Set Disk Cache Policy to Disabled to avoid data loss in the event of an unexpected power-off.</p> <p>openGauss supports using an NVMe SSD with the SAS API deployed in RAID mode as the primary storage device of the database.</p>
Network requirements	<p>300 Mbit/s or faster Ethernet.</p> <p>You are advised to bind two NICs for redundancy.</p>

Software Requirements and Dependencies

Software requirements

Software Type	Configuration
Linux operating system (OS)	<ul style="list-style-type: none">ARM: openEuler 20.3 LTS (recommended); Kirin V10x86: openEuler 20.3 LTS; CentOS 7.6 <p>Note: Ensure that the OS language is set to English, or the installation package cannot be installed properly.</p>
Linux file system	It is recommended that there be at least 1.5 billion available iNodes remaining.
Tools	bzip2
Python	<ul style="list-style-type: none">openEuler: Python 3.7.XCentOS: Python 3.6.XKirin: Python 3.7.X <p>Note: Python needs to be built using --enable-shared.</p>

Dependencies

Software	Recommended Version
libaio-devel	0.3.109-13
flex	2.5.31 or later
bison	2.7-4
ncurses-devel	5.9-13.20130511
glibc-devel	2.17-111
patch	2.7.1-10
lsb_release	4.1
readline-devel	7.0-13
libnsl (in the openEuler + x86 environment)	2.28-36

Modifying OS Configurations (1)

- **Disable the OS firewall** to ensure that the openGauss can be used properly when the firewall is enabled.

- Step 1: Change the value of **SELINUX** in the **/etc/selinux/config** file to **disabled**.

- 1. Use **vim** to open the **config** file.

```
vim /etc/selinux/config
```

- 2. Change the value of SELINUX to disabled and enter :wq to save the change and exit.

```
SELINUX=disabled
```

- Step 2: Reboot the system.

```
reboot
```

- Step 3: Check whether the firewall is disabled.

```
systemctl status firewalld
```

- If the firewall status is **active (running)**, the firewall is not disabled. Go to step 4.

- If the firewall status is **inactive (dead)**, the firewall is disabled. Skip step 4.

- Step 4: Disable the firewall.

```
systemctl disable firewalld.service  
systemctl stop firewalld.service
```

- Step 5: Repeat step 1 to step 4 on each host.

Modifying OS Configurations (2)

- Set the character set parameters.

- Set the same character set for all database nodes. You can add **export LANG=Unicode** to the **/etc/profile** file.

```
vim /etc/profile  
export LANG=en_US.UTF-8
```

- Set the time zone and time.

- Ensure that the time zone and time on each database node are consistent.
 - Step 1: Run the following command to check whether the time and time zone of each database node are consistent: If they are not, perform steps 2 to 3.

```
date
```

- Step 2: Run the following command to copy the **/etc/localtime** file to the **/usr/share/zoneinfo/** directory of each database node:

```
cp /usr/share/zoneinfo/$Locale/$Time zone /etc/localtime
```

- Step 3: Set the time of each database node to the same time. For example:

```
date -s "Sat Sep 27 16:00:07 CST 2020"
```

Modifying OS Configurations (3)

- **Disable RemoveIPC.**
 - On each database node, disable RemoveIPC. In CentOS 7.6, RemoveIPC is disabled by default, and you can skip this step.
 - Step 1: Change the value of **RemoveIPC** in the **/etc/systemd/logind.conf** file to **no**.
 - Use **vim** to open the **logind.conf** file.
 - Change the value of **RemoveIPC** to **no**.
 - Step 2: Change the value of **RemoveIPC** in the **/usr/lib/systemd/system/systemd-logind.service** file to **no**.
 - Use **vim** to open the **systemd-logind.service** file.
 - Change the value of **RemoveIPC** to **no**.
 - Step 3: Reload the configuration parameters.
`systemctl daemon-reload
systemctl restart systemd-logind`
 - Step 4: Check whether the modification takes effect.
`loginctl show-session | grep RemoveIPC
systemctl show systemd-logind | grep RemoveIPC`
 - Step 5: Repeat steps 1 to 4 on each host.

```
vim /etc/systemd/logind.conf
```

```
RemoveIPC=no
```

```
vim /usr/lib/systemd/system/systemd-logind.service
```

```
RemoveIPC=no
```

Modifying OS Configurations (4)

- Set the NIC MTU.

- Set the NIC MTU on each database node to the same value.
- Step 1: Query the NIC name of the server: **ifconfig**
- In the example shown here, the server IP address is **10.244.53.173**, and the NIC name for the server is **eth0**.

```
[root@pekpopgsci00181 ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.244.53.173 netmask 255.255.254.0 broadcast 10.244.53.255
        inet6 fe80::f816:3eff:fe0c:3 prefixlen 64 scopeid 0x20<link>
            ether fa:16:3e:f4:e0:c3 txqueuelen 1000 (Ethernet)
            RX packets 99430 bytes 86571697 (82.5 MiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 72881 bytes 6419720 (6.1 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.31.8.93 netmask 255.255.240.0 broadcast 172.31.15.255
        inet6 fe80::f816:3eff:fe1c:7244 prefixlen 64 scopeid 0x20<link>
            ether fa:16:3e:1c:72:44 txqueuelen 1000 (Ethernet)
            RX packets 9408 bytes 2123616 (2.0 MiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 38 bytes 5793 (5.6 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- Step 2: Set the NIC MTU on each database node to the same value. For x86, the recommended MTU value is **1500**. For ARM, the recommended MTU value is **8192**.

```
ifconfig N/C name mtu value
```

Enabling Remote Login for **root**

- During openGauss installation, the **root** account is required for remote login. This section describes how to enable remote login for **root**.
- Step 1: Modify the **PermitRootLogin** configuration to enable remote login as **root**.

- Open the **sshd_config** file.
- Modify permissions for **root** using either of the following methods:

- Comment out **PermitRootLogin no**.
- Change the value of **PermitRootLogin yes**.
- Run the **:wq** command to save the modification and exit.

```
vim /etc/ssh/sshd_config
```

```
#PermitRootLogin no
```

```
PermitRootLogin yes
```

- Step 2: Modify the **Banner** configuration to delete the welcome information displayed when you connect to the system. The welcome information affects remote operations during the installation.

- Edit the **sshd_config** file.
- Comment out the line where **Banner** is.
- Run **:wq** to save the modification and exit.

```
vim /etc/ssh/sshd_config
```

```
#Banner XXXX
```

- Step 3: Restart sshd to make the settings take effect:
- Step 4: Log in to the system as user **root** again.

```
systemctl restart sshd.service
```

```
ssh xxx.xxx.xxx.xxx
```

Installation User and User Group

- To minimize the account permissions during the installation and ensure system security of openGauss after the installation is complete, the installation script automatically creates a database installation user and this user will be used as the administrator for subsequent operations and maintenance of the openGauss.

User/Group Name	Type	Planning Suggestions
dbgrp	OS	<p>Plan a separate user group, for example, dbgrp. The database installation user will belong to this group. This user group is specified by the -G parameter during installation environment initialization. If this user group does not exist, the installation script automatically creates it. Alternatively, you can create a user group before the installation by running groupadd dbgrp. User permissions are checked when the gs_preinstall script is executed. The gs_preinstall script automatically grants access and execution permissions for the installation directory and data directory to all users in this user group.</p>
omm	OS	<p>You are advised to plan the user omm for openGauss operation and maintenance. The user omm is the OS user specified by the -U parameter during initialization of the installation environment. If this user already exists, delete it or change the initial user. For security purposes, the group for this user is dbgrp.</p>

- During openGauss installation, when **gs_install** is executed, a database user **omm**, with the same name as the installation user is created. This user has the highest operation permissions on the database. The initial password of this user is specified by the user.

Installing openGauss – Installing openGauss on a Single Node (1)

- **Prerequisites**

- A user group and a common user have been created.
- All the server OSs and networks are functioning properly.
- Common users must have the read, write, and execute permissions on the database package decompression directory and installation directory, and the installation directory must be empty.
- Common users have the execution permission for the downloaded openGauss package.
- Before the installation, check whether the specified openGauss port is occupied. If the port is occupied, change the port number or stop the process that uses the port.

- **Procedure**

- Step 1: Log in to the host where the openGauss package is installed as a common user and decompress the openGauss package to the installation directory.

```
tar -jxf openGauss-x.x.x-openEuler-64bit.tar.bz2 -C /opt/software/openGauss
```

- Step 2: Assuming the decompressed package is stored in `/opt/software/openGauss`, go to the `simpleInstall` directory.

```
cd /opt/software/openGauss/simpleInstall
```

- Step 3: Run the `install.sh` script to install openGauss.

```
sh install.sh -w xxxx
```

Installing openGauss – Installing openGauss on a Single Node (2)

- **Procedure**

- Step 4: After the installation is complete, use **ps** and **gs_ctl** to check whether the process is normal.

Running **ps** command should display information similar to the following:

```
>> ps ux | grep gaussdb
omm 24209 11.9 1.0 1852000 355816 pts/0 Sl 01:54 0:33 /opt/software/openGauss/bin/gaussdb -D
/opt/software/openGauss/single_node
omm 20377 0.0 0.0 119880 1216 pts/0 S+ 15:37 0:00 grep --color=auto gaussdb
```

Running **gs_ctl** should display information similar to the following:

```
>> gs_ctl query -D /opt/software/openGauss/data/single_node
gs_ctl query ,datadir is /opt/software/openGauss/data/single_node
HA state:
local_role : Normal
static_connections : 0
db_state : Normal
detail_information : Normal
```

Senders info:
No information

Receiver info:
No information

Installing openGauss – Installing openGauss on One Primary Node and One Standby Node (1)

- Prerequisites

- A user group and a common user have been created.
- All the server OSs and networks are functioning properly.
- Common users must have the read, write, and execute permissions on the database package decompression directory and installation directory, and the installation directory must be empty.
- Common users must have the execution permission on the downloaded openGauss package.
- Before the installation, check whether all ports in the specified openGauss port matrix are occupied. If they are occupied, change the ports or stop the processes that use the ports. For details about the port number, see the parameter description in step 3.

- Procedure

- Step 1: Log in to the host where the openGauss package is installed as a common user and decompress the package to the installation directory.

```
tar -jxf openGauss-x.x.x-openEuler-64bit.tar.bz2 -C /opt/software/openGauss
```

- Step 2: Assuming that the decompressed package is stored in the **/opt/software/openGauss** directory, go to the **simpleInstall** directory.

```
cd /opt/software/openGauss/simpleInstall
```

- Step 3: Run the **install.sh** script to install openGauss.

```
sh install.sh -w xxxx --multinode
```

Installing openGauss – Installing openGauss on One Primary Node and One Standby Node (2)

- Procedure

- Step 4: After the installation is complete, use **ps** and **gs_ctl** to check whether the process is normal.

Running **ps** should display information similar to the following:

```
>> ps ux | grep gaussdb
omm 4879 11.8 1.1 2082452 373832 pts/0 Sl 14:26 8:29 /opt/software/openGauss/bin/gaussdb -D
/opt/software/openGauss/data/master -M primary

omm 5083 1.1 0.9 1819988 327200 pts/0 Sl 14:26 0:49 /opt/software/openGauss/bin/gaussdb -D
/opt/software/openGauss/data/slave -M standby

omm 20377 0.0 0.0 119880 1216 pts/0 S+ 15:37 0:00 grep --color=auto gaussdb
```

Installing openGauss – Installing openGauss on One Primary Node and One Standby Node (3)

- Procedure

- Step 5: After the installation is complete, use **ps** and **gs_ctl** to check whether the process is normal. Running **gs_ctl** should display information similar to the right:

```
>> gs_ctl query -D /opt/software/openGauss/data/master
gs_ctl query ,datadir is /opt/software/openGauss/data/master
HA state:
local_role          : Primary
static_connections   : 1
db_state             : Normal
detail_information   : Normal

Senders info:
sender_pid           : 5165
local_role            : Primary
peer_role             : Standby
peer_state            : Normal
state                 : Streaming
sender_sent_location  : 0/4005148
sender_write_location : 0/4005148
sender_flush_location : 0/4005148
sender_replay_location: 0/4005148
receiver_received_location: 0/4005148
receiver_write_location: 0/4005148
receiver_flush_location: 0/4005148
receiver_replay_location: 0/4005148
sync_percent          : 100%
sync_state             : Sync
sync_priority          : 1
sync_most_available    : Off
channel               : 10.244.44.52:27001-->10.244.44.52:35912

Receiver info:
No information
```

Starting and Stopping openGauss

- Starting openGauss

- Step 1: Log in to the primary database node as **omm**.
 - Step 2: Start openGauss:

```
gs_om -t start
```

Start openGauss:

```
>> gs_om -t start  
Starting cluster.
```

```
=====  
=====  
Successfully started.
```

- Stopping openGauss

- Step 1: Log in to the primary database node as **omm**.
 - Step 2: Stop openGauss:

```
gs_om -t stop
```

Stop openGauss:

```
>> gs_om -t stop  
Stopping cluster.  
=====  
Successfully stopped cluster.  
=====  
End stop cluster.
```

Contents

1. openGauss Installation
- 2. Connection and Authentication**
3. openGauss Tools
4. Database Uninstallation

Setting the Client Authentication Policy

- The current default values for parameters in the openGauss configuration file (**pg_hba.conf**) are all for standalone deployment. Applications can set parameters as needed by invoking **gs_guc**. For more details, see *Developer Guide*. Set the client authentication policy and send semaphores to the database process.

```
gs_guc [ set | reload ] [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -h "HOSTTYPE DATABASE  
USERNAME IPADDR-WITH-IPMASK AUTHMEHOD authentication-options"
```

- set**

Modifies only parameters in the configuration file.

- reload**

Modifies the parameters in the configuration file and sends semaphores to the database process for reloading the configuration file.

- N**

Specifies the name of the host to be set.

Value: the name of the existing host

When this parameter is set to **ALL**, all the hosts in openGauss are configured.

- I INSTANCE-NAME**

Specifies the name of the instance to be configured.

Value: the name of the existing instance

When this parameter is set to **ALL**, all the instances in the host are to be set.

- D**

Specifies the openGauss instance directory where the command is run. When the **encrypt** command is used, this parameter indicates the directory where the generated password file is stored.

- h host-auth-policy**

Specifies the client authentication policy added to the **pg_hba.conf** configuration file. Values supported:

- HOSTTYPE DATABASE USERNAME IPADDR IPMASK [authentication-options]
- HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK [authentication-options]
- HOSTTYPE DATABASE USERNAME HOSTNAME [authentication-options]

The **HOSTTYPE** parameter is mandatory and can be set to any of the following values:

- Local
- Host
- Hostssl
- hostnossal

local is a Unix domain socket. **host** is a common or SSL-encrypted TCP/IP socket. **hostssl** is an SSL-encrypted TCP/IP socket. **hostnossal** is a TCP/IP-only socket. The **authentication-options** parameter is optional and can be set to any of the following values:

- trust
- reject
- md5
- sha256
- cert
- gss

For details about the parameters, see their description in the **pg_hba.conf** configuration file.

Setting Parameters in the Configuration File

- The current default values for parameters in the openGauss configuration file (**postgresql.conf**) are all for standalone deployment. Applications can set parameters as needed by invoking **gs_guc**.
- Modifying parameters in the configuration file (**postgresql.conf**).

```
gs_guc set [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -c  
"parameter = value"
```

- Resetting parameters to their default values.

```
gs_guc [ set | reload ] [-N NODE-NAME] [-I INSTANCE-NAME | -D  
DATADIR] -c "parameter"
```

Example 1: You can modify database node parameters. For example, you could change the maximum number of connections allowed by a database to **800**. The database must be restarted for the change take effect.

```
gs_guc set -N all -I all -c "max_connections = 800"
```

```
Total instances: 21. Failed instances: 0.  
Success to perform gs_guc!
```

- **set**

Modifies only parameters in the configuration file.

- **reload**

Modifies the parameters in the configuration file and sends semaphores to the database process for reloading the configuration file.

- **-N**

Specifies the name of the host to be configured.

Value: the name of the existing host

When this parameter is set to **ALL**, all the hosts in openGauss are configured.

- **-I INSTANCE-NAME**

Specifies the name of the instance to be configured.

Value: the name of the existing instance

When this parameter is set to **ALL**, all the instances in the host are configured.

- **-D**

Specifies the openGauss instance directory where the command is run. When the **encrypt** command is used, this parameter indicates the directory for storing the generated password file.

- **-c parameter=value**

Specifies the name and value of the openGauss configuration parameter to be set.

Value range: all the parameters in the **postgresql.conf** file

- **-U, --keyuser=USER**

gsql Client Connection – Confirming the Connection Information

- The client tool connects to the database through the primary database node. Before connecting to the database, you need the IP address and port for the server where the primary database node is located.
- Step 1: Log in to the primary database node as user **omm**.
- Step 2: Run the **gs_om-t status--detail** command to query instances in openGauss.

```
gs_om -t status --detail
[ DBnode State ]

node      node_ip      instance          state
-----
1  plat1 192.168.0.11  5001 /srv/BigData/gaussdb/data1/dbnode Normal
```

- In the command output shown here, **192.168.10.11** is the IP address of the server where the primary database node instance is deployed. The data directory of the primary database node is **/srv/gaussdb/data1/dbnode**.
- Step 3: Check the port for the primary database node. Check the port in the **postgresql.conf** file in the data directory of the primary database node obtained in step 2. For example:

```
cat /srv/BigData/gaussdb/data1/dbnode/postgresql.conf | grep port
port = 8000  # (change requires restart)
#comm_sctp_port = 1024  # Assigned by installation (change requires restart)
#comm_control_port = 10001  # Assigned by installation (change requires restart)
# supported by the operating system:
# e.g. 'localhost=10.145.130.2 localport=12211 remotehost=10.145.130.3 remoteport=12212, localhost=10.145.133.2
localport=12213 remotehost=10.145.133.3 remoteport=12214'
# e.g. 'localhost=10.145.130.2 localport=12311 remotehost=10.145.130.4 remoteport=12312, localhost=10.145.133.2
localport=12313 remotehost=10.145.133.4 remoteport=12314'
# %r = remote host and port
alarm_report_interval = 10
support_extended_features=true
```

Connecting to a Database Locally Using gsql (1)

- gsql is a CLI database connection tool provided by openGauss. gsql provides both basic and more advanced database functions to facilitate user operations. By default, if a client idle after connecting to a database, the client automatically disconnects from the database after a period specified by **session_timeout**. To disable session timeouts, set **session_timeout** to 0.
- Step 1: Log in to the primary database node as **omm**.
- Step 2: Connect to the database. After the database is installed, the **postgres** database is generated by default. The first time you connect, you can connect to this database.

Run the following command to connect to the database:

```
gsql -d postgres -p 8000
```

If information similar to the following is displayed, the connection was successful:

```
gsql ((openGauss 2.0.0 build 78689da9) compiled at 2021-03-31 21:03:52 commit 0 last mr )
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
```

```
postgres=#
```

If you log in and connect to the database as administrator **omm**, **DBNAME=#** is displayed. If you log in and connect to the database as a common user, **DBNAME=>** is displayed. **Non-SSL connection** indicates that the database connection is not using SSL. For improved security, you are advised to connect to the database using SSL.

Connecting to a Database Locally Using gssql (2)

- Step 3: Change the password after your first login. (You do not need to change the password in openGauss 2.0.0 and later versions.) The initial password is set manually during openGauss database installation. For details, see "Installing the openGauss > Executing Installation" in the *Installation Guide*. You need to change the password during initial login. You can use the following command:

```
postgres=# ALTER ROLE omm IDENTIFIED BY 'newpassword' REPLACE 'XuanYuan@2012';
```

- Step 4: Exit the database.

```
postgres=# \q
```

Using gsql to Remotely Connect to a Database - Configuring a Whitelist Using gs_guc

- Step 1: Log to the primary database node as **omm**.
- Step 2: Configure the client authentication mode and enable the client to connect to the host as **jack**. User **omm** cannot be used for remote connection. Assume a client whose IP address is 10.10.0.30 is allowed to access the local host.

```
gs_guc set -N all -I all -h "host all jack 10.10.0.30/32 sha256"
```

This command adds a rule to the **pg_hba.conf** file corresponds to the primary database node for authenticating clients connected to the primary database node. Each record in the **pg_hba.conf** file can be in one of the following four formats. For parameter description of the four formats, see the *Configuration File*.

```
local  DATABASE USER METHOD [OPTIONS]
host   DATABASE USER ADDRESS METHOD [OPTIONS]
hostssl DATABASE USER ADDRESS METHOD [OPTIONS]
hostnossal DATABASE USER ADDRESS METHOD [OPTIONS]
```

During authentication, the system checks connection requests with records in the **pg_hba.conf** file in sequence, so the record sequence is important.

- The suggestions on configuring authentication rules are as follows:
 - Records placed at the front have strict connection parameters but weak authentication methods.
 - Records placed at the end have weak connection parameters but strict authentication methods.

Installing the gsql Client and Connecting to a Database (1)

- On the client host, upload the client tool package and configure environment variables for the gsql client.
- Step 1: Log in to the client as **root**.
- Step 2: Create the **/tmp/tools** directory.
mkdir /tmp/tools
- Step 3: Obtain the **openGauss-1.1.0-openEuler-64bit-Libpq.tar.gz** package from the software installation package and upload it to the **/tmp/tools** directory.
- Step 4: Decompress the file.
cd /tmp/tools
tar -zxvf openGauss-1.1.0-openEuler-64bit-Libpq.tar.gz
- Step 5: Log in to the server where the primary database node is located, copy the **bin** directory in the database installation directory to the **/tmp/tools** directory on the client host, and then log in to the client host to perform step 6.

```
scp -r /opt/huawei/install/app/bin root@10.10.0.30:/tmp/tools
```

In the preceding command, **/opt/huawei/install/app** is the **{gaussdbAppPath}** from the **cluster_config.xml** file, and **10.10.0.30** is the IP address of the client host.

- Step 6: Log in to the host where the client is installed and set the environmental variables in the **~/.bashrc** file.

```
vi ~/.bashrc
```

Enter the following information and run **:wq!** to save and exit.

```
export PATH=/tmp/tools/bin:$PATH  
export LD_LIBRARY_PATH=/tmp/tools/lib:$LD_LIBRARY_PATH
```

Installing the gsql Client and Connecting to a Database (2)

- Step 7: Apply the environmental variables.

```
source ~/.bashrc
```

- Step 8: Connect to the database. After the database is installed, a **postgres** database is generated by default. The first time you connect, you can connect to this database.

```
gsql -d postgres -h 10.10.0.11 -U jack -p 8000 -W Test@123
```

- **postgres** is the name of the database, **10.10.0.11** is the IP address of the server where the primary database node resides, **jack** is the user of the database, **8000** is the port number of the primary database node, and **Test@123** is the password of user **jack**.

Overview of System Catalogs and System Views

- System catalogs store structured metadata of openGauss. They are the source of information used by openGauss to control system operations and are a core component of the database system.
- System views provide ways to query the system catalogs and internal database status.
- System catalogs and system views are visible to either system administrators or all users.
- You can delete and re-create system catalogs, add columns to them, and insert and update values in them, but doing so may make system information inconsistent and cause system faults. Generally, users should not modify system catalogs or system views, or rename their schemas. They are automatically maintained by the system.
- **Notice:** Do not add, delete, or modify system catalogs because doing so will result in exceptions or even openGauss unavailability.

Contents

1. openGauss Installation
2. Connection and Authentication
- 3. openGauss Tools**
4. Database Uninstallation

openGauss Tools

Category	Tool	Description
Client tools	gsql	gsql is a CLI database connection tool provided by openGauss. Users can use gsql to connect to a server and perform operations and maintenance on the server. In addition to basic functions for performing operations on a database, gsql provides several other more advanced features.
Server tools	gs_check	gs_check has been enhanced, incorporating various check tools, such as gs_check and gs_checkos. It helps users check all sorts of items, such as openGauss runtime, the OS, network, and database environments, and can perform comprehensive environmental checks before major operations in openGauss. gs_check helps ensure smoother operations.
	gs_checkos	gs_checkos checks OS version information, control parameters, and disk configurations, and configures control parameters, I/O parameters, network parameters, and THP services.
	gs_collector	If there is a problem with openGauss, gs_collector collects OS, log, and configuration file information to make troubleshooting easier.
	gs_dump	gs_dump exports database information. It can export complete, consistent data of database objects (such as databases, schemas, tables, and views) without affecting normal access for database users.
	gs_dumpall	gs_dumpall exports database information. It can export complete and consistent data of the database without affecting normal access for database users.
	gs_guc	gs_guc sets parameters in the openGauss configuration file (postgresql.conf or pg_hba.conf). The default parameter values in the configuration file are for standalone deployment, but you can modify parameter values as needed by invoking gs_guc.
	gs_om	openGauss provides gs_om to help you maintain openGauss, including starting openGauss, stopping openGauss, querying openGauss status, generating static configuration files, updating dynamic configuration files, replacing SSL certificates, starting and stopping Kerberos authentication, and displaying help and version information.
	gs_restore	gs_restore, provided by openGauss, is used to import data that was exported using gs_dump . It can also be used to import files exported by gs_dump .
	gs_ssh	gs_ssh, provided by openGauss, helps users run the same command on multiple nodes in openGauss.

Client Tools – gsSQL

- After a database is deployed, you need certain tools to connect to the database for operations and commissioning. openGauss provides some convenient tools to help you connect to your database. You can use these tools to easily connect to a database and perform various operations.
- gsSQL is a CLI database connection tool provided by openGauss. Users can use gsSQL to connect to a server and perform operations and maintenance on the server. In addition to basic functions for performing operations on a database, gsSQL also provides several more advanced features.
- Basic functions:
 - **Connect to the database.** For details, see section "Connection and Authentication."
 - **Run SQL statements.** Interactively entered SQL statements and specified SQL statements in a file can be run.
 - **Run meta-commands.** Meta-commands help the administrator view database object information, query cache information, format SQL output, and connect to a new database.

Serveaxr Tools

- During the use of openGauss, installation, upgrade, and health management are **all involved**. To facilitate openGauss maintenance, openGauss provides a set of management tools.
 - gs_check
 - gs_checkos
 - gs_checkperf
 - gs_clean
 - gs_collector
 - gs_dump
 - gs_dumpall
 - gs_guc
 - gs_om
 - gs_restore
 - gs_ssh

Server Tools – gs_check (1)

- gs_check has been improved, incorporating various check tools, such as gs_check and gs_checkos. These tools allow you to check openGauss runtime; the OS, network, and database running environments, and they let you perform comprehensive environmental checks before major operations in openGauss, ensuring smoother database operations.
- Notes:
 - The **-i** or **-e** arguments must be used. **-i** specifies a single item to be checked, and **-e** specifies an inspection scenario where multiple items will be checked.
 - If **-i** is not set to a **root** item or no such items are contained in the checklist of the scenario specified by **-e**, you do not need to enter the name or password of user **root**.
 - You can run **--skip-root-items** to skip **root** items.
 - If the MTU values are inconsistent, the check may be slow or check process may become unresponsive. If the inspection tool displays a message, change the MTU values of the nodes to be the same and then repeat the inspection.
 - If the switch does not support the configured MTU value, process response failures may be caused by communication problems even if the MTU values are the same. In this case, you need to adjust the MTU based on the switch.

Server Tools – gs_check (2)

- Syntax:

- Perform a single-item check:

```
gs_check -i ITEM [...] [-U USER] [-L] [-I LOGFILE] [-o OUTPUTDIR] [--skip-root-items][--set][--routing]
```

- Perform a scenario-specific check:

```
gs_check -e SCENE_NAME [-U USER] [-L] [-I LOGFILE] [-o OUTPUTDIR] [--skip-root-items] [--time-out=SECS][--set][--routing][--skip-items]
```

- Display help information:

```
gs_check -? | --help
```

- Display version information.

```
gs_check -V | --version
```

Server Tools – gs_checkos

- gs_checkos helps you check the OS version, various control parameters and disk configurations, and it lets you configure control parameters, I/O parameters, network parameters, and THP services.
- Prerequisites:
 - The hardware and network are working properly.
 - Among the hosts, the mutual trust relationships for **root** are normal.
 - Only **root** is authorized to run the **gs_checkos** command.
- Syntax:
 - Check OS information. `gs_checkos -i ITEM [-f HOSTFILE] [-h HOSTNAME] [-X XMLFILE] [--detail] [-o OUTPUT] [-I LOGFILE]`
 - Display the help information. `gs_checkos -? | --help`
 - Display the version information. `gs_checkos -V | --version`

Server Tools – gs_checkperf (1)

- openGauss provides the gs_checkperf tool for you to routinely check the openGauss-level performance (for instance, host CPU usage, GaussDB CPU usage, and I/O usage), host-level performance (CPU, memory, and I/O usage), session-/process-level performance (CPU, memory, and I/O usage), and SSD performance (write and read performance). This helps you track the load on openGauss and fine-tune database performance accordingly.
- Prerequisites:
 - openGauss is running properly and is not in read-only mode.
 - Services on the database are running normally.
- Note:
 - The monitoring information of gs_checkperf comes from tables in PMK mode. If no ANALYZE operation is performed on such tables, gs_checkperf execution may fail. Typical error information is as follows:

LOG: Statistics in some tables or columns(pmk.pmk_snapshot.snapshot_id) are not collected.

▫ To fix this error, HINT: Do analyze for them in order to generate optimized plan.

```
analyze pmk.pmk_configuration;
analyze pmk.pmk_meta_data;
analyze pmk.pmk_snapshot;
analyze pmk.pmk_snapshot_dbnode_stat;
analyze pmk.pmk_snapshot_datanode_stat;
```

Server Tools – gs_checkperf (2)

- Syntax:

- Check the SSD performance (as **root**).

```
gs_checkperf -U USER [-o OUTPUT] -i SSD [-I LOGFILE]
```

- Check the openGauss performance (as the openGauss installation user).

```
gs_checkperf [-U USER] [-o OUTPUT] [-i PMK] [--detail] [-I LOGFILE]
```

- Display the help information.

```
gs_checkperf -? | --help
```

- Display the version information.

```
gs_checkperf -V | --version
```

Server Tools – gs_checkperf (3)

Category	Performance Parameter	Description
openGauss level	Host CPU usage	CPU usage of the host
	GaussDB CPU usage	CPU usage of the GaussDB
	Shared memory hit rate	Hit rate of the shared memory
	In-memory sort ratio	Ratio of completed sorts in memory
	I/O usage	Number and time of file reads and writes
	Disk usage	Number of file writes, average write time, and maximum write time
	Transaction statistics	Number of current SQL executions and sessions
Host level	CPU usage	Host CPU usage, including CPU busy time and CPU idle time
	Memory usage	Host memory usage, including total physical memory and used memory
	I/O usage	Number and time of file reads and writes
Session/Process level	CPU usage	Session CPU usage, including CPU busy time and CPU idle time
	Memory usage	Session memory usage, including total physical memory and used memory
	I/O usage	Number of shared buffer hits in a session
SSD performance (Only root can view it.)	Write performance	The dd command (flag=direct bs=8M count=2560) is used to write data into an SSD every 10 seconds.
	Read performance	The dd command (flag=direct bs=8M count=2560) is used to read data from an SSD every 7s.

Server Tools – gs_clean

- If a database breaks down unexpectedly, there may be residual temporary tables left over. `gs_clean` can clear out these tables.
- Syntax: Connect to the primary DN and clear out any residual temporary tables from transactions on the primary DN.

```
gs_clean [OPTION ...] [USERNAME]
```

- Parameter description: The values of **OPTION** are as follows:

- `-a, --all`
Clears all residual temporary tables in available databases.
- `-h, --host=HOSTADDRESS`
Specifies the IP address of the host where the target DN is located.
Value range: IP address of the host where the target DN is located
Default value: **localhost**
- `-p, --port=PORT`
Specifies the port number of the primary DN.
Value range: port number of the target DN
Default value: **5432** If the environment variable *PGPORT* of the OS has been set, the value of *PGPORT* is used by default.
- `-q, --quiet`
Prints only error information in quiet mode.
- `-r, --rollback`
Rolls back all abnormal transactions.
- `-t, --timeout=SECS`
Specifies the timeout mechanism.
Default value: **5**, in seconds.

- `-U, --username=USERNAME`
Specifies the name of the user for connecting to the database.
Value range: a valid username in the database
Default value: *OS username*
- `-v, --verbose`
Outputs more detailed restoration information.
- `-V, --version`
Outputs version information about this tool.
- `-w, --no-password`
Specifies that no password authentication is required.
- `-W, --password=PASSWORD`
Specifies that password authentication is required.
Value range: a valid string. The password must meet complexity requirements.
- `-e, --exclusive`
Only deletes temporary tables.
- `-j, --jobs`
Specifies the number of concurrent tasks for clearing out two-phase residual files.
Value range: 1–10
The default value is **3**.
- `-?, --help`
Displays the help information.

Server Tools – gs_collector

- If openGauss becomes faulty, gs_collector helps with troubleshooting by collecting logs, configuration files, and information about the OS. You can use the **-C** parameter to specify the information to be collected.
- Prerequisites: An OS tool (for example, gstack) that gs_collector depends on has been installed. If it is not installed, an error message is displayed, and corresponding collection item is skipped.
- Syntax:

- Collect logs (as a non-root user).

```
gs_collector --begin-time= "BEGINTIME" --end-time= "ENDTIME" [-h HOSTNAME | -f HOSTFILE] [--keyword=KEYWORD] [--speed-limit=SPEED]
[-o OUTPUT] [-I LOGFILE] [-C CONFIGFILE]
```

- Display the help information.

```
gs_collector -? | --help
```

- Display the version information.

```
gs_collector -V | --version
```

Server Tools – gs_dump (1)

- gs_dump, provided by openGauss, is used to export database information. You can export a database or its objects (such as schemas, tables, and views). The database can be the default **postgres** database or a user-specified database.
- gs_dump is executed by the OS user **omm**.
- When gs_dump is used to export data, other users can still access (read and write) the openGauss database.
- gs_dump can export complete and consistent data. For example, if gs_dump has started exporting database A at T1, data of the database at T1 will be exported, but any modifications to the database after T1 will not be exported.
- gs_dump can export database information as a plain-text SQL script or as an archive file.
 - Plain-text SQL script: It contains the SQL statements required to restore the database. You can use gsql to execute the SQL script. With only minor modifications, the SQL script can rebuild a database on other hosts or database products.
 - Archive file: It contains data required to restore the database. It can be a .tar file, a directory, or a custom format. For details, see the table on the next slide. The export result must be used with gs_restore to restore the database. The system allows users to select or even to sort the content to be imported.

Server Tools – gs_dump (2)

- gs_dump can create four export file formats, which are specified by the **-F** or **--format** option. The following table lists the file formats.

Format	Value of -F	Description	Suggestion	Import Tool
Plain-text	p	A plain-text script file containing SQL statements and commands. The commands can be executed on gsql, a command line terminal, to recreate database objects and load table data.	You are advised to use plain-text export files for small databases.	Before using gsql to restore database objects, you can use a text editor to edit the file as required.
Custom	c	A binary file that enables the restoration of all or just selected database objects from an exported file.	You are advised to use custom archive files for medium or large databases.	You can use gs_restore to import database objects from an exported custom archive file.
Directory	d	A directory containing directory files and the data files of tables and BLOBS.	-	
.tar	t	A .tar archive file that allows for the restoration of all or selected database objects from an exported file. It cannot be further compressed and has an 8-GB limitation on the size of a single table.	-	

- Syntax:

```
gs_dump [OPTION]... [DBNAME]
```

Server Tools – gs_dumpall (1)

- `gs_dumpall`, provided by openGauss, is used to export all openGauss database information, including data of the default database **postgres**, user-defined databases, and common global objects of all openGauss databases.
- `gs_dumpall` is executed by the OS user **omm**.
- When `gs_dumpall` is used to export data, other users can still access (read and write) databases.
- `gs_dumpall` can export complete and consistent data. For example, if `gs_dumpall` is enabled to export openGauss database at T1, the openGauss data status at the time point T1 is exported, but modifications to openGauss after T1 will not be exported.
- `gs_dumpall` exports all databases in two parts:
 - `gs_dumpall` exports all global objects, including information about database users and groups, tablespaces, and attributes (for example, global access permissions).
 - `gs_dumpall` calls `gs_dump` to export SQL scripts from each openGauss database, which contain all the SQL statements required to restore databases.
- The exported files are both plain-text SQL scripts. Use `gsql` to execute them to restore openGauss databases.

Server Tools – gs_dumpall (2)

- Note:
 - Do not modify any exported file or content, or restoration may fail.
 - To ensure the data consistency and integrity, gs_dumpall acquires a shared lock on the table to be dumped. If another transaction has acquired a shared lock on the table, gs_dumpall waits until this lock is released and then locks the table for dumping. If the table cannot be locked in the specified time, the dump will fail. You can customize the timeout interval to wait for lock releasing by specifying the **--lock-wait-timeout** parameter.
 - During an export, gs_dumpall reads all the tables in a database, so you need to connect to the database as an openGauss administrator to export a complete file. When you use gsql to execute SQL scripts, administrator permissions are also required to add users and user groups, and create databases.
- Syntax:

```
gs_dumpall [OPTION]...
```

Server Tools – gs_guc (1)

- The current default values for parameters in the openGauss configuration file (**postgresql.conf** or **pg_hba.conf**) are all for standalone deployment. Applications can set parameters as needed by invoking **gs_guc**. **gs_guc** is executed by OS user **omm**.
- **gs_guc-current.log**
 - This file records the logs generated by **gs_guc**.
 - Default directory: **\$GAUSSLOG/bin/gs_guc**
- **gs_guc- Year-Month-Day_HHMMSS.log**
 - A backup file is generated based on the current time when the size of the **gs_guc-current.log** file reaches 16 MB.
- **server.key.cipher, server.key.rand**
 - When you encrypt a user password using the **gs_guc encrypt** command and the **-M** option is set to **server**, the **server.key.cipher** and **server.key.rand** files are generated. **server.key.cipher** stores the encrypted password. **server.key.rand** stores the encryption factor.
- **client.key.cipher, client.key.rand**
 - When you encrypt a user password using the **gs_guc encrypt** command and the **-M** option is set to **client**, the **client.key.cipher** and **client.key.rand** files are generated. **client.key.cipher** stores the encrypted password. **client.key.rand** stores the encryption factor.
- **datasource.key.cipher, datasource.key.rand**
 - When you encrypt a user password using the **gs_guc encrypt** command and the **-M** option is set to **source**, the **datasource.key.cipher** and **datasource.key.rand** files are generated. **datasource.key.cipher** stores the encrypted password. **datasource.key.rand** stores the encryption factor.

Server Tools – gs_guc (2)

- Syntax:

- Check parameters in the configuration file.

```
gs_guc check [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -c "parameter"
```

- Modify parameters in the configuration file.

```
gs_guc set [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -c "parameter = value"
```

- Reset parameters to their default values.

```
gs_guc [ set | reload ] [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -c "parameter"
```

- Modify parameters in the configuration file and send semaphores to the **postgresql.conf** configuration file.

```
gs_guc reload [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -c parameter=value
```

- Modify the client authentication policy and send semaphores to the **pg_hba.conf** configuration file.

```
gs_guc [ set | reload ] [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -h "HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK  
AUTHMETHOD authentication-options"
```

- Comment out a configured client authentication policy and send semaphores to **pg_hba.conf**.

```
gs_guc [ set | reload ] [-N NODE-NAME] [-I INSTANCE-NAME | -D DATADIR] -h "HOSTTYPE DATABASE USERNAME IPADDR-WITH-IPMASK  
AUTHMETHOD"
```

Server Tools – gs_om (1)

- openGauss provides the gs_om tool to help maintain the database, including starting and stopping openGauss, querying openGauss status, querying static configurations, generating static configuration files, querying the openGauss status details, generating the dynamic configuration file, replacing the SSL certificate, and displaying the help information and version number. gs_om is executed by OS user **omm**.
 - Start openGauss.

```
▫ gs_om -t start [-h HOSTNAME] [-D dataDir] [--time-out=SECS] [--security-mode=MODE] [-I LOGFILE]
```

```
▫ gs_om -t stop [-h HOSTNAME] [-D dataDir] [--time-out=SECS] [-m MODE] [-I LOGFILE]
```

```
▫ gs_om -t restart [-h HOSTNAME] [-D dataDir] [--time-out=SECS] [--security-mode=MODE] [-I LOGFILE] [-m MODE]
```

```
gs_om -t status [-h HOSTNAME] [-o OUTPUT] [--detail] [--all] [-I LOGFILE]
```

Server Tools – gs_om (2)

- Generate a static configuration file.

```
gs_om -t generateconf -X XMLFILE [--distribute] [-I LOGFILE]
```

- Generate a dynamic configuration file. Perform this operation after a failover or switchover from the standby node to the primary node.

```
gs_om -t refreshconf
```

- Display the static configurations.

```
gs_om -t view [-o OUTPUT]
```

- Query openGauss status details.

```
gs_om -t query [-o OUTPUT]
```

- Replace the SSL certificate.

```
gs_om -t cert --cert-file=CERTFILE [-I LOGFILE]
```

```
gs_om -t cert --rollback
```

- Enable or disable the Kerberos authentication in the database.

```
gs_om -t kerberos -m [install|uninstall] -U USER [-I LOGFILE] [--krb-client|--krb-server]
```

- Display the help information.

```
gs_om -? | --help
```

Server Tools – gs_restore

- gs_restore, provided by openGauss, is used to import data or files that were exported using gs_dump.
- gs_restore is executed by OS user **omm**.
- It has the following functions:
 - Import data to a database.
 - If a database is specified, data is imported to the database. For parallel import, the database password is required.
 - Create a script for database recreation.
 - If a database is not specified for storing the imported data, a script containing the SQL statement needed to recreate the database is created and written to a file or standard output. This script output is equivalent to the plain text output used by gs_dump.
- Syntax:

```
gs_restore [OPTION]... FILE
```

Server Tools – gs_ssh

- gs_ssh, provided by openGauss, helps users run the same command on multiple nodes in openGauss.
- Note:
 - gs_ssh can run only commands that the current database user has permissions to run.
 - The commands executed by gs_ssh do not affect the current session. For example, the **cd** or **source** command affects only the process environment. It does not affect the current session environment.
- Prerequisites:
 - The trust relationships between the hosts are normal.
 - openGauss has been installed and deployed properly.
 - The command to be run can be found by the **which** command, and the current user has the required execution permissions.
 - gs_ssh is executed by OS user **omm**.
- Syntax:
 - Run commands synchronously. `gs_ssh -c cmd`
 - Display the help information. `gs_ssh -? | --help`
 - Display the version information. `gs_ssh -V | --version`

Contents

1. openGauss Installation
2. Connection and Authentication
3. openGauss Tools
- 4. Database Uninstallation**

Uninstalling the openGauss (1)

- openGauss provides an uninstallation script.
- The process of uninstalling openGauss includes uninstalling openGauss and deleting the environment of the openGauss server.
- Procedure:
 - Step 1: Log in to the primary database node as **omm**.
 - Step 2: Use **gs_uninstall** to uninstall openGauss.

A **gs_uninstall --delete-data** command on the primary database node.

- Troubleshooting:
 - If the uninstallation fails, troubleshoot the issue by following the log information provided in the **\$GAUSSLOG/om/gs_uninstall-YYYY-MM-DD_HHMMSS.log** file.

Execute the **gs_uninstall** script to uninstall the openGauss.

```
gs_uninstall --delete-data
Checking uninstallation.
Successfully checked uninstallation.
Stopping the cluster.
Successfully stopped the cluster.
Successfully deleted instances.
Uninstalling application.
Successfully uninstalled application.
Uninstallation succeeded.
```

Execute the **gs_uninstall** script to perform single-node uninstallation.

```
gs_uninstall --delete-data
Checking uninstallation.
Successfully checked uninstallation.
Stopping the cluster.
Successfully stopped the cluster.
Successfully deleted instances.
Uninstalling application.
Successfully uninstalled application.
Uninstallation succeeded.
```

Uninstalling openGauss (2)

- **One-click environment cleanup:** After openGauss is uninstalled, if you do plan on deploying openGauss again, you can run the `gs_postuninstall` script to clean up the environment information from the openGauss server. The `gs_postuninstall` script is used to clean up the environment settings made by the `gs_preinstall` script.
- **Prerequisites:**
 - openGauss has already been uninstalled.
 - Mutual trust has been established between different **root** users.
 - Only **root** is authorized to run the `gs_postuninstall` command.
- **Procedure:**
 - Step 1: Log in to the openGauss server as **root**.
 - Step 2: Check whether mutual trust has been established between different **root** users. If it has not, establish mutual trust manually. Run `ssh {hostname}` to check if the mutual trust relationships have been successfully established. Then, enter `exit`.

```
plat1:~ # ssh plat2
Last login: Tue Jan  5 10:28:18 2016 from plat1
Huawei's internal systems must only be used for conducting Huawei's business or for purposes authorized by Huawei
management. Use is subject to audit at any time by Huawei management.
plat2:~ # exit
logout
Connection to plat2 closed.
plat1:~ #
```

Quiz

1. (Single-choice) Which of the following is not an openGauss server tool?
 - A. gs_check
 - B. gs_guc
 - C. gsql
 - D. gs_collector
2. (Single-choice) When the gs_checkperf tool is used to check system performance, which of the following can be viewed only by user **root**?
 - A. openGauss level
 - B. Host level
 - C. Session/Process level
 - D. SSD performance

Quiz

3. (Multiple-choice) Which of the following are the prerequisites for installing openGauss on a single node?
- A. User groups and common users have been created, and the operating systems and networks of all servers are running properly.
 - B. Common users must have the read, write, and execute permissions on the database package decompression directory and installation directory, and the installation directory must be empty.
 - C. A common user has the execution permission on the downloaded openGauss package.
 - D. Before the installation, check whether the specified openGauss port is occupied. If the port is occupied, change the port number or stop the process that uses the port.

Summary

- This chapter describes how to install, deploy, connect, authenticate, and uninstall the openGauss database, as well as the functions and usage of common tools provided by openGauss.

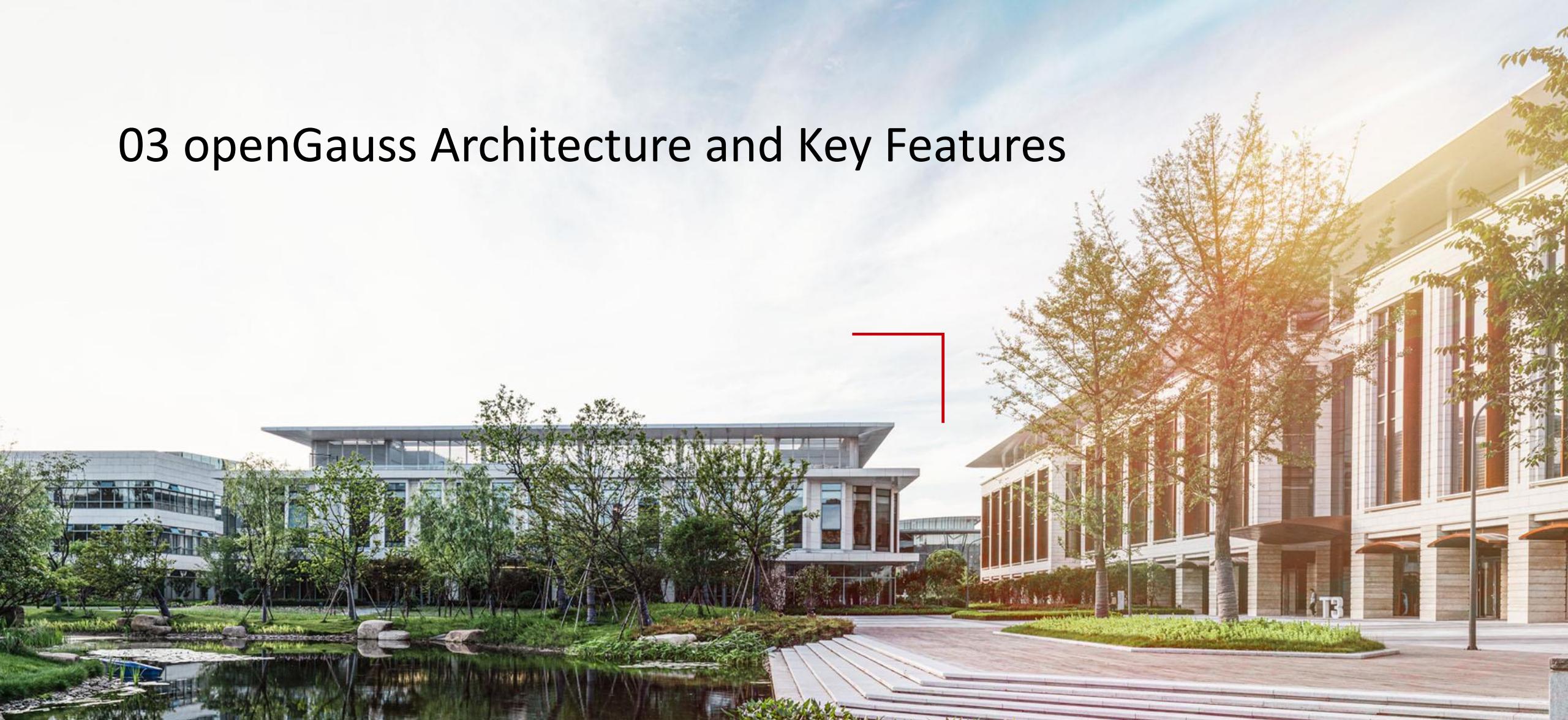
Recommendations

- HUAWEI CLOUD:
 - <https://www.huaweicloud.com/>
- openGauss help document:
 - <https://opengauss.org/zh/docs/2.0.0/docs/installation/installation.html>
- openGauss code repository:
 - <https://gitee.com/opengauss>

Acronyms and Abbreviations

Acronym and Abbreviation	Full Spelling
CLI	Command line interface
MTU	Maximum transmission unit
RAID	Redundant array of independent disks
XML	Extensible Markup Language

03 openGauss Architecture and Key Features



Foreword

- This chapter describes the physical and logical architecture of the openGauss database, along with the typical networking and deployment scheme for openGauss databases. In addition, this chapter describes the key features of openGauss database in terms of high performance, high availability, high security, easy maintenance, and AI capabilities.

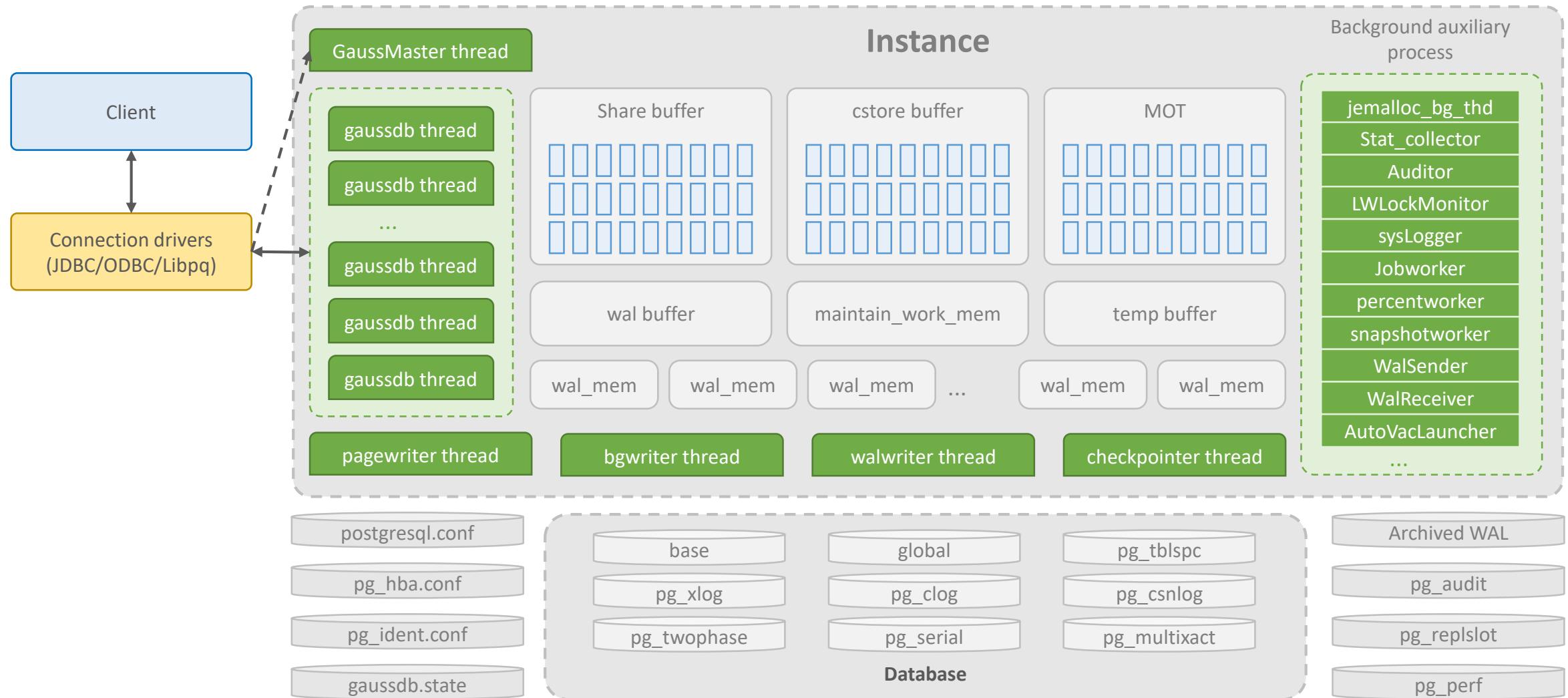
Objectives

- Upon completing this course, you will be able to:
 - Be familiar with the physical and logical openGauss architectures.
 - Learn typical openGauss networking and deployment solutions.
 - Understand the key features of openGauss databases.

Contents

- 1. System Architectures**
2. Deployment Solutions
3. Typical Network Topology
4. Key Features

openGauss Architecture



Memory Structure (1)

- **share buffer**: the shared memory buffer of the database server. Read and write operations in the database system are performed on the data in the memory. The data in the disk must be loaded to the memory (the database cache) before being processed. The memory is used as a bridge between a slow disk and a fast CPU to accelerate the I/O access speed.
- **cstore buffer**: shared buffer used by column stores. In scenarios where column-store tables are mainly used, the shared buffer is rarely used. In this scenario, you should reduce the value of **shared_buffers** and increase the value of **cstore_buffers**.
- **MOT**: memory-optimized table. All data and indexes are stored in the memory. MOT has significant advantages in terms of performance (query and transaction latency), scalability (throughput and concurrency), and, in some cases, even costs (high resource utilization).

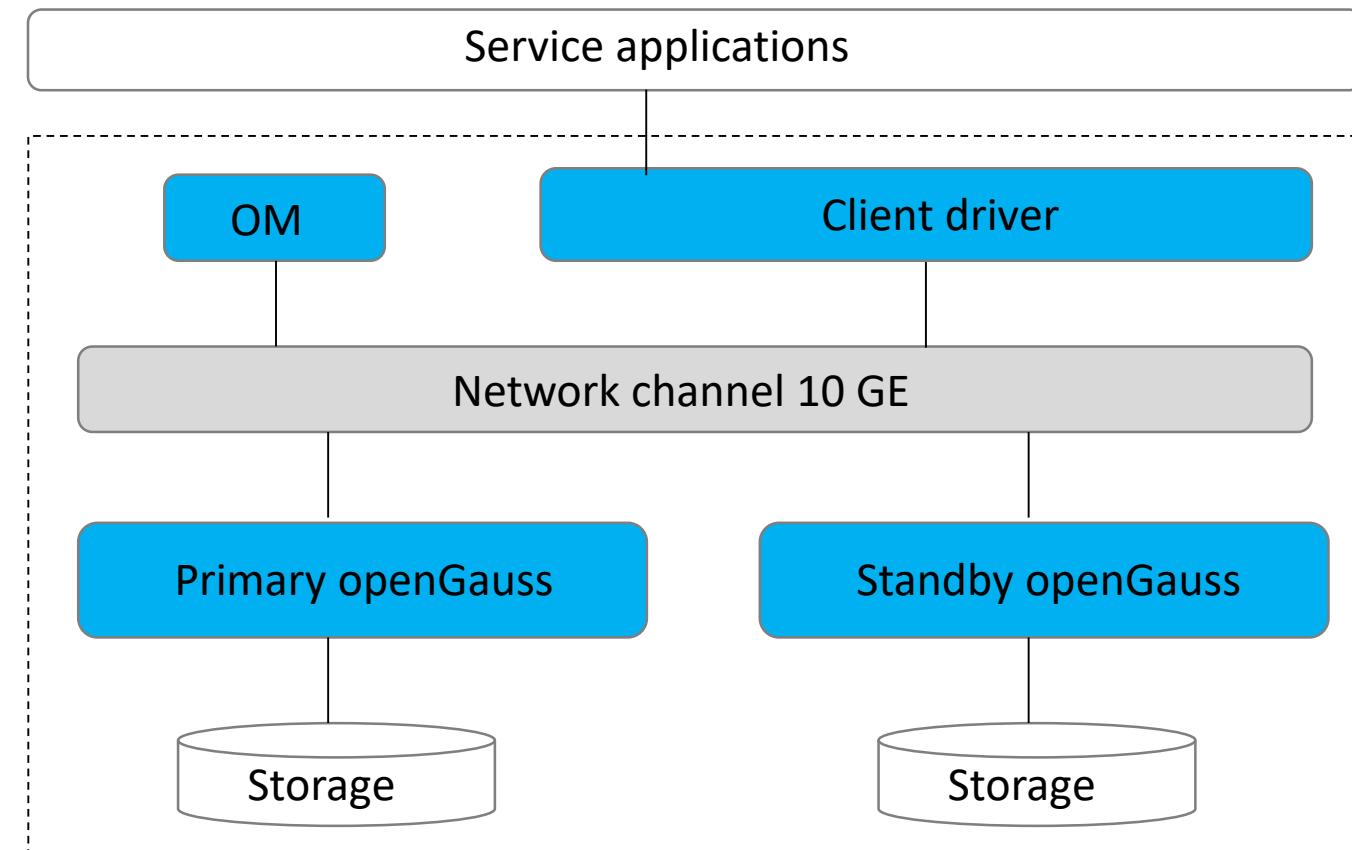
Memory Structure (2)

- **wal buffer**: the shared memory used by WALs that have not been written into disks.
- **maintain_work_mem**: local memory used for maintenance operations such as VACUUM, index creation, and table modification.
- **work_mem**: used for query operations, such as sorting or hash tables.
- **temp buffer**: a temporary buffer used by database sessions to access temporary tables.

Main Threads

- **GaussMaster**: the openGauss management thread, which is also called the postmaster thread. It manages the database, such as starting and stopping the database and forwarding messages.
- **pagewriter**: flushes dirty page data from memory to disks.
- **bgwriter**: copies dirty page data to the double-writer area, writes the data to disks, and forwards the dirty pages to the bgwriter subthread to write the data to disks.
- **walwriter**: updates the WAL page data in the memory to the WAL file to ensure that all submitted transactions are permanently recorded and will not be lost.
- **checkpoint**: triggered periodically. Each time the checkpoint thread is triggered, all dirty pages are flushed to the disk.

openGauss Logical Architecture

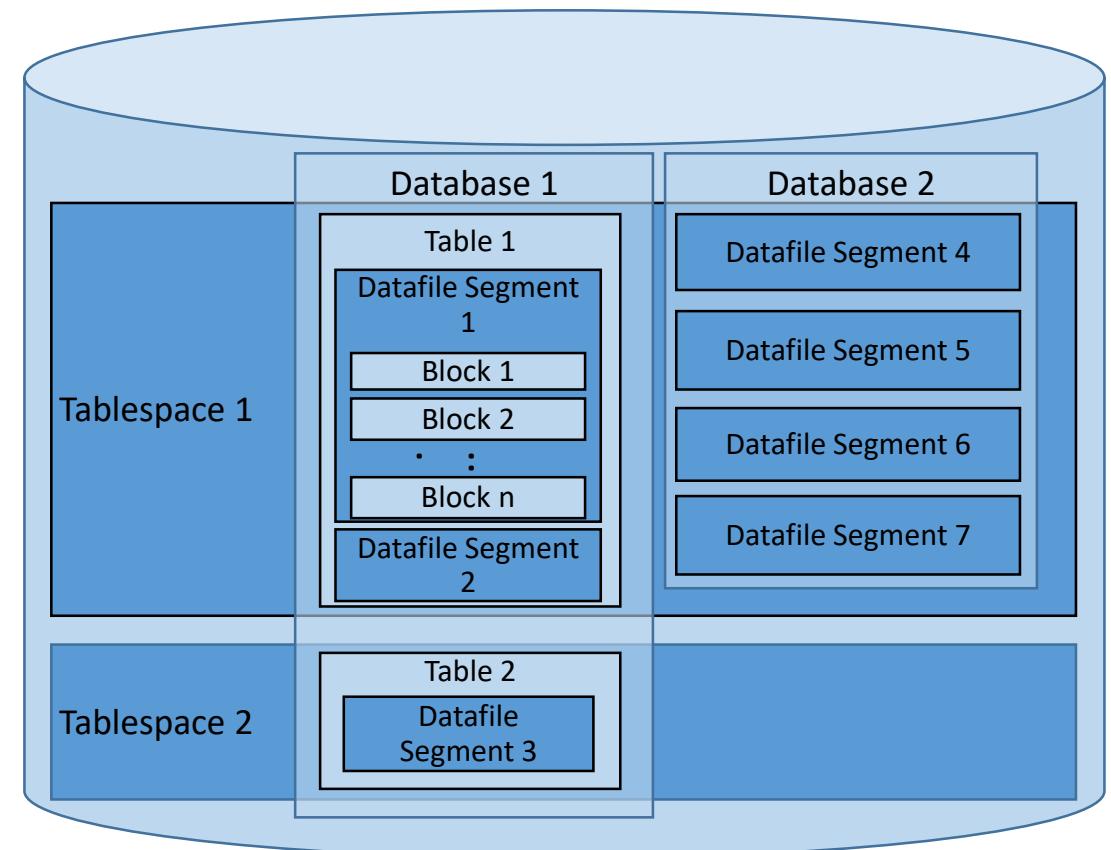


Logical Architecture Description

Software	Configuration
OM	The operation manager (OM) provides management APIs and tools for routine maintenance and database configuration.
Client driver	The client driver receives the access requests from applications, and returns the execution results to the applications. It communicates with openGauss instances, issues application SQL commands, and receives openGauss execution results.
openGauss (primary/standby)	Data nodes (DNs) store service data, execute data queries, and return execution results to the client. openGauss supports one primary and multiple standby instances. Primary and standby openGauss instances are deployed on different physical nodes.
Storage	Functions as the server's local storage resources to store data permanently.

Database Logical Architecture

- DNs in openGauss store data on disks. This section describes the logical objects on each DN and the relationships between these objects.
- Database logical architecture:
 - Tablespace: Directories storing the actual files of the databases. Multiple tablespaces can coexist, and each of them can contain files belonging to different databases.
 - Database: A database manages various data objects. Databases are isolated from each other. Objects managed by a database can be distributed to multiple tablespaces.
 - Datafile Segment: A data file that stores data of only one table. A table containing more than 1 GB of data is stored in multiple datafile segments.
 - Table: One table belongs to only one database and one tablespace. The datafile segments storing the data of the same table must be in the same tablespace.
 - Block: Basic unit of database management. Its default size is 8 KB.



Database Directory Structure (1)

- By default, database files are stored in the data directory created during database initialization (command: **initdb**). There are many types of directories and files with different functions in the data directory. In addition to data files, there are parameter files, control files, database run logs, and write-ahead logs.
- Partial data directory structure:

Directory/File	Parent Directory	Function
bin	-	Saves binary files of the database.
lib	-	Saves library files of the database.
share	-	Saves common files required for the database to run, such as the configuration file template.
data (database node/primary database node)	-	Stores database node data. For the primary node, the directory name is data_dnxxx . For the standby node, the directory name is data_dnSxxx . xxx indicates the database node number.
base	Instance data directory	Contains subdirectories of each database.

Database Directory Structure (2)

- Partial data directory structure:

Directory/File	Parent Directory	Function
global	Instance data directory	Contains the subdirectories of the cluster range table.
pg_audit	Instance data directory (configurable)	Stores database audit logs.
pg_log	Instance data directory (configurable)	Stores the run logs of database node instances.
pg_xlog	Instance data directory	Stores write-ahead logs.
postgresql.conf	Instance data directory	Parameter file
pg_hba.conf	Instance data directory	Client authentication control file
postmaster.opts	Instance data directory	Records the command line parameters used during server startup.

Database Directory Structure (3)

- Partial data directory structure:

Directory/File	Parent Directory	Function
gs_initdb	bin	Database initialization tool
gs_dump	bin	Exports database information.
gs_ctl	bin	Database service control tool
gs_guc	bin	You can set application parameters by using gs_guc .
gsql	bin	Database connection tool running in the CLI

Contents

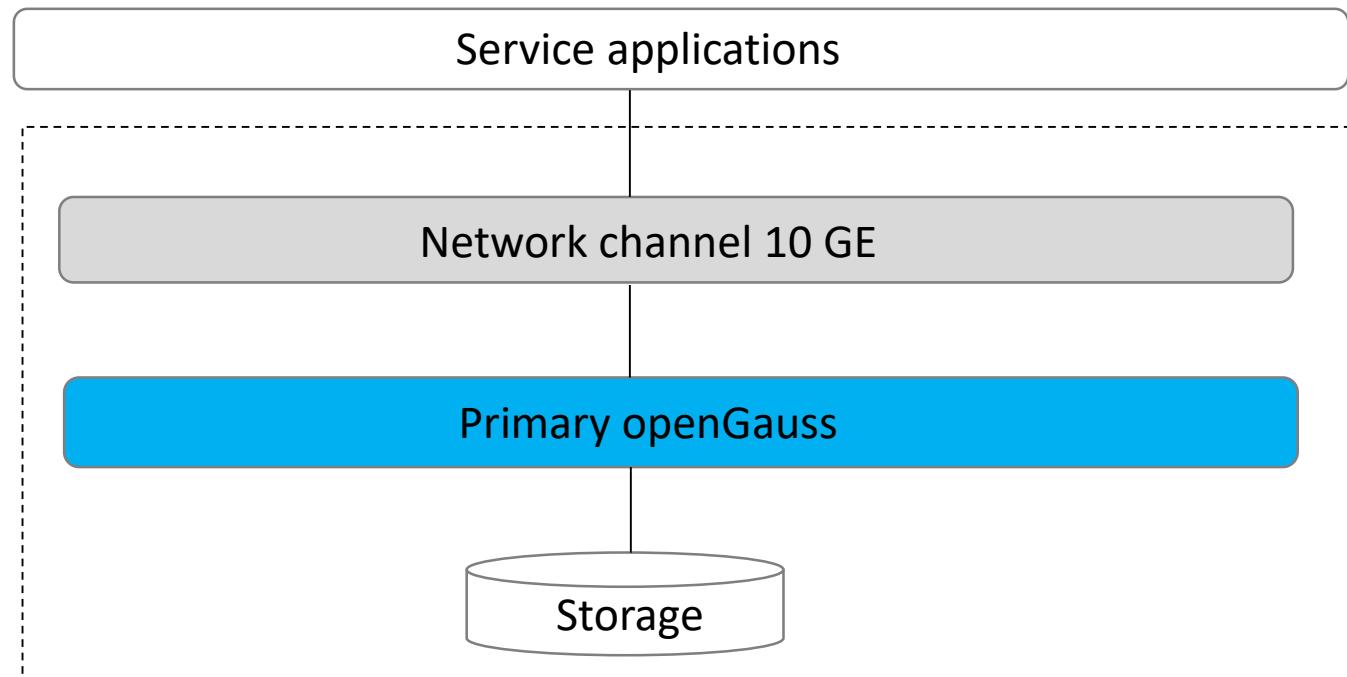
1. System Architectures
- 2. Deployment Solutions**
3. Typical Network Topology
4. Key Features

Introduction to Deployment Solutions

Deployment	Solution	HA	Basic Configuration Requirement	Service Scenario	Feature	Technical Specifications
Standalone	Standalone	HA is not supported.	Single equipment room	Physical machine	No reliability and availability requirements for the system. Suitable for trial use and commissioning scenarios.	RTO and RPO are uncontrollable. Instance-level DR is not supported. If faults occur, the system can become unavailable. Lost instance data cannot be restored.
Primary/Standby	Primary and standby nodes	Instance faults can be withstood.	Single equipment room	Physical machine	There is no network delay between nodes. Instance faults in the database can be withstood. Suitable for scenarios with low reliability requirements.	RPO=0 Instance fault RTO < 10s AZ-level DR is not supported. Primary and standby nodes configured for maximum availability are recommended.
One primary and multiple standbys	One primary and multiple standbys (Quorum)	Instance faults can be withstood.	Single equipment room	Physical machine	There is no network delay between nodes. Instance faults in the database can be withstood.	RPO=0 Instance fault RTO < 10s AZ-level DR is not supported. Primary/standby synchronization is recommended. 2 to 4 copies.

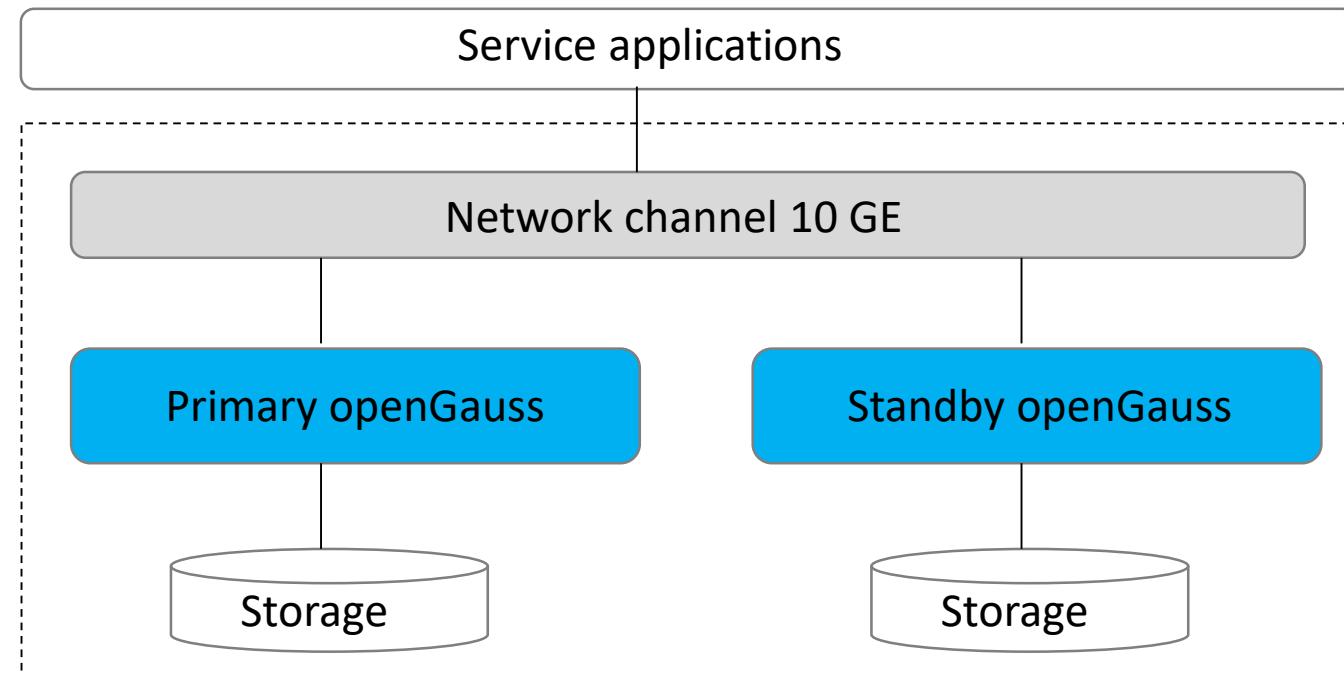
Standalone Deployment

- Standalone deployment does not ensure reliability or availability. There is only one data copy. Once data is damaged or lost, only physical backups can be used to restore data. Therefore, this deployment mode is only recommended for scenarios such as just trying out a database or commissioning syntax functions in a test environment. You are not advised to use this mode on a live commercial network.

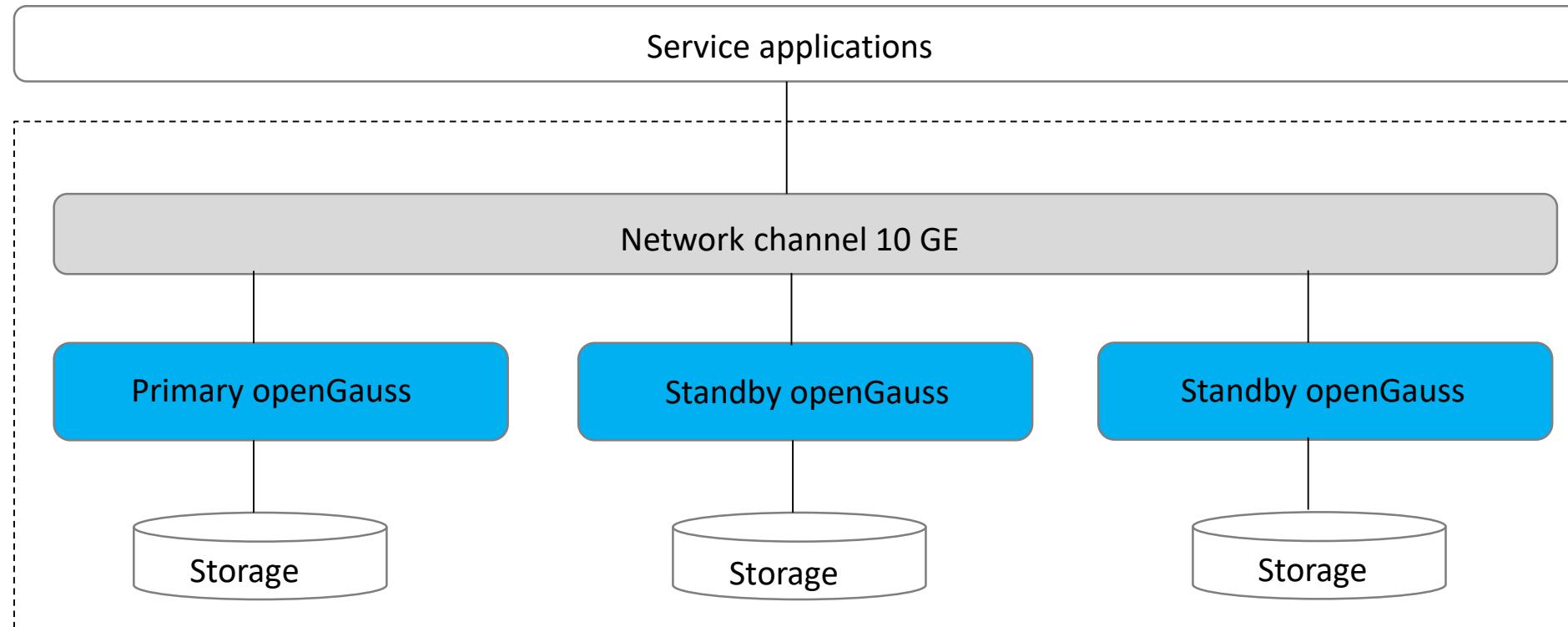


Primary/Standby Deployment

- The primary/standby deployment means there are two data copies, one for the primary node and the other for the standby node. The standby node receives and plays back the logs.



One Primary and Multiple Standbys Deployment



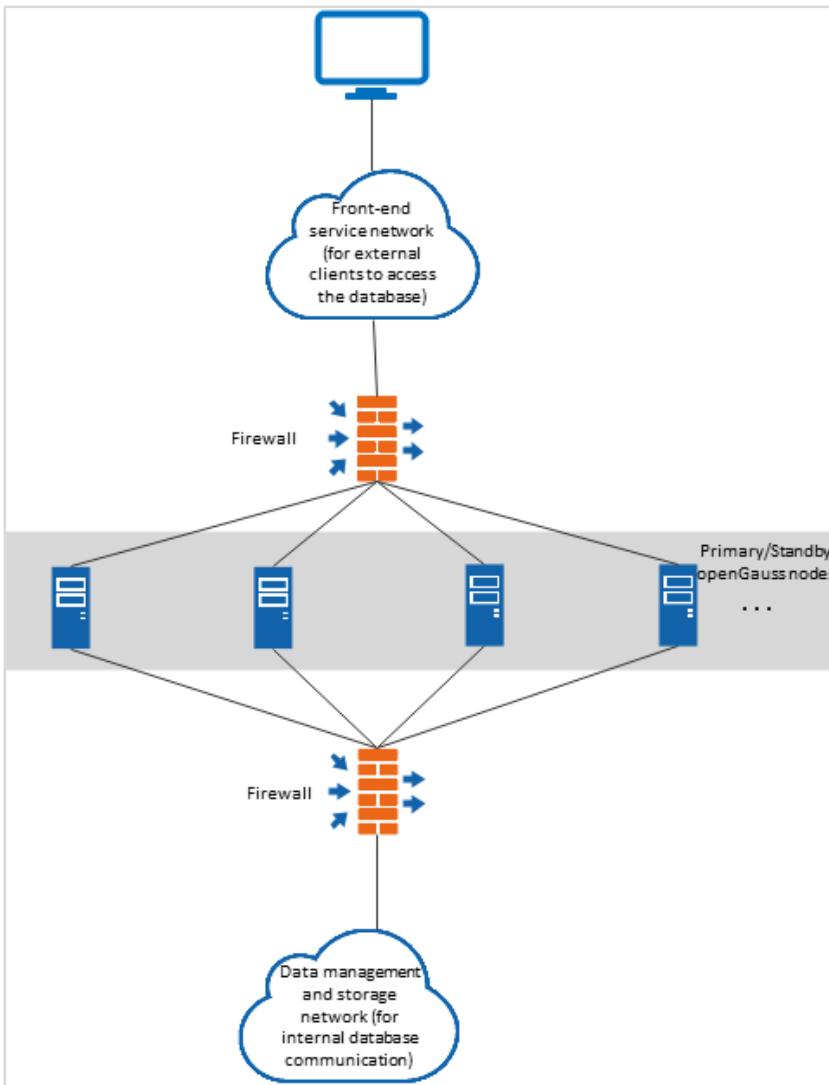
One Primary and Multiple Standbys Deployment

- Multi-copy deployment provides the capability of defending against instance-level faults. This mode is appropriate when equipment room DR is not required but some hardware faults need to be protected against.
- Generally, multi-copy deployment means one primary and two standbys, so there are three copies total. This deployment mode provides 99.99% reliability, which is good enough for most applications.
 - In primary/standby quorum replication, data is synchronized to at least one standby node. Additional nodes can be used, but using a single standby node ensures the best performance.
 - If any of the primary or standby nodes fails, services will not be affected.
 - There are three copies of the data. If one node fails, the system still has two copies of the data in reserve, and any standby DN can be promoted to primary.
 - The primary and standby nodes cannot be deployed on the same physical machine.

Contents

1. System Architectures
2. Deployment Solutions
- 3. Typical Network Topology**
4. Key Features

Typical Network Topology

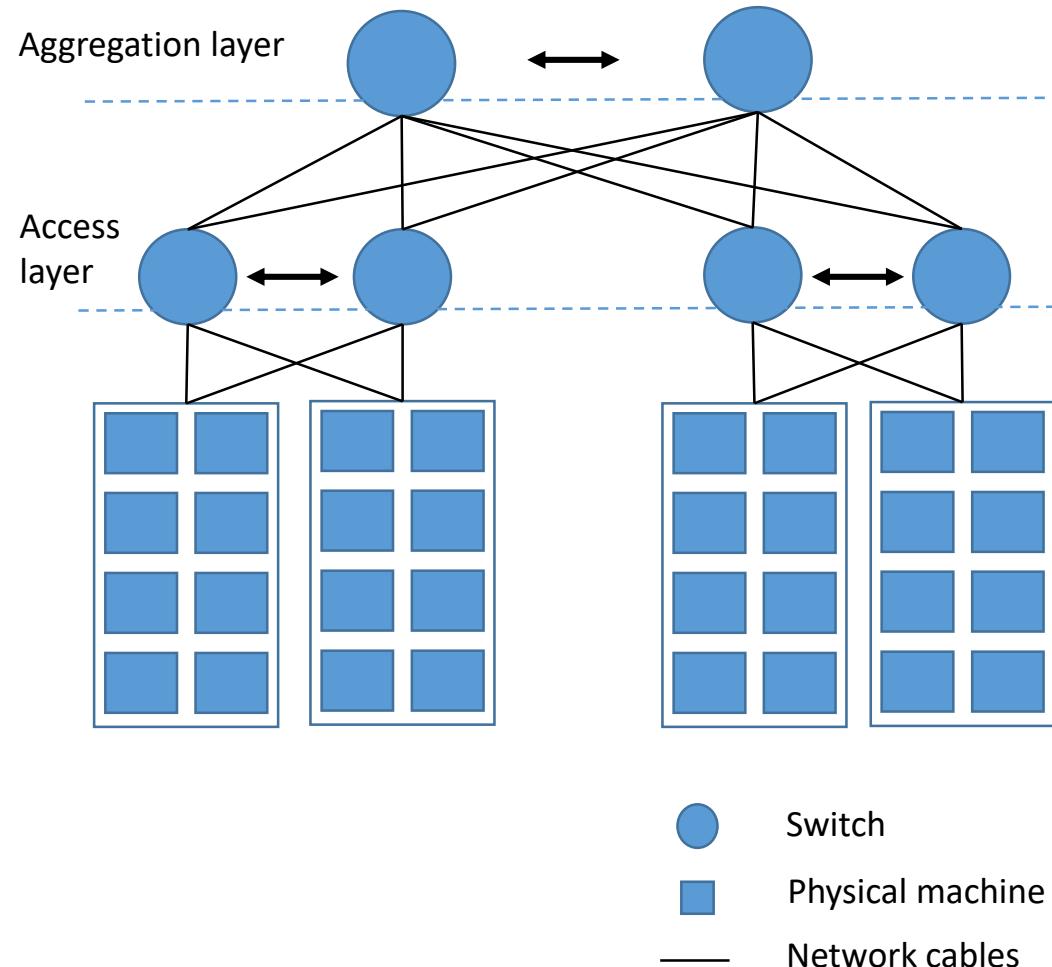


- Network Division
 - Database management and storage network: Through this network, DBAs call OM scripts to manage and maintain openGauss instances. It is also used for openGauss primary/standby communication networking. The database management and storage network are also used for applications monitoring the system.
 - Frontend service network: allows external clients to access the openGauss database.

Typical Network Topology

- The typical network topology has the following advantages:
 - The service network is isolated from the database management and storage network, effectively protecting the security of backend storage data.
 - The isolation between the service network and database management and storage network improves system security, preventing attackers from managing database servers over the Internet.
- Network exclusiveness and 1:1 bandwidth convergence ratio are the basic requirements for openGauss database network performance. In a production system, the backend storage network needs to be reserved exclusively for data transport, and it must have a 1:1 bandwidth convergence ratio.

Database Management and the Storage Network



- To achieve a convergence ratio of 1:1, the bandwidth doubles each time the switching network layer is increased by one layer.
- In the figure, each solid line (the network cables) indicates 80GE bandwidth, that is, the sum of the bandwidth upper limits of eight physical machines.
- At the access layer, each switch provides 160GE downlink bandwidth and 160GE uplink bandwidth. The convergence ratio is 1:1.
- The access bandwidth of each switch at the aggregation layer is 320GE.

Contents

1. System Architectures
2. Deployment Solutions
3. Typical Networking
- 4. Key Features**

- High Performance
 - High Availability (HA)
 - High Security
 - Easy Maintenance
 - AI Capabilities

Optimizers

- Optimizers are built in databases and are used to generate the best possible execution plans in the SQL parsing phase. Optimizers use either rule-based optimization (RBO) or cost-based optimization (CBO).
 - RBO optimizers
 - RBO optimizers generate execution plans based on the table structure and SQL text and are not affected by the table data.
 - CBO optimizers
 - CBO optimizers generate execution plans based on the statistics.

RBO

- Rule-based optimization
 - An optimizer selects an execution plan based on available access paths and their levels. That is, the optimizer has several embedded rules, and the execution plan is generated based on the rule that the executed SQL statements comply with.
- There are the following types of rules:
 - Index selection
 - Sorting elimination and MIN/MAX optimization
 - Join sequence selection
 - Sub-query rewriting
 - Condition rewriting

CBO (1)

- A CBO consists of three important components:
 - Query converter
 - For some statements, the query converter rewrites a SQL statement into an equivalent SQL statement with a lower cost and determines whether the conversion is worth performing.
 - Cost estimator
 - It is used to determine the overall cost of a given execution plan.
 - Execution plan generator
 - It explores various SQL execution plans by trying different access paths, join methods, and join orders.

CBO (2)

- The CBO estimates execution costs based on statistics. By using the CBO, the database calculates the number of tuples and the execution cost for each execution step under each execution plan based on the number of table tuples, column width, NULL record ratio, and characteristic values, such as distinct and MCV values, and certain cost calculation methods. The database then selects the execution plan that incurs **the lowest costs** for the overall execution or for the return of the first tuple.
- When using a CBO, ensure that sufficient statistics are collected for tables and related indexes. You are advised to periodically analyze tables and indexes that are frequently added, deleted, or modified.
- Only when the database has sufficient statistical data to reflect the actual situation can it make the right choice.

Statistics

- What Is Statistics Information
 - The CBO needs to select the data query mode based on the data in the table or index. Because the table contains a large amount of data, the data volume and data distribution in the statistics table cannot be collected in real time during each query. Therefore, data needs to be periodically analyzed and the data distribution of the table and index needs to be saved to the data dictionary for the optimizer to use. This is the statistics.
- openGauss generates execution plans based on cost estimates and optimized for lower costs. Optimizers need to estimate the number of data rows and the cost based on statistics collected using **ANALYZE**. The statistics are vital for the estimation of the number of rows and cost. Global statistics are collected using **ANALYZE: relpages** and **reltuples** in the **pg_class** table; **stadistinct**, **stanullfrac**, **stanumbersN**, **stavalueN**, and **histogram_bounds** in the **pg_statistic** table.

When to Collect Statistics

- After executing batch insertions and deletions, you are advised to run the **ANALYZE** statement on the table or the entire library to update statistics.
- For an intermediate table generated during the execution of a batch script or stored procedure, you also need to run the **ANALYZE** statement.
- If there are multiple inter-related columns in a table and the conditions or grouping operations based on these columns are involved in the query, collect statistics about these columns so that the query optimizer can accurately estimate the number of rows and generate an effective execution plan.

How to Collect Statistics

- The **ANALYZE** statement collects statistics on database table contents. These statistics will be stored in the **PG_STATISTIC** system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.
- Procedure

- Update statistics on a single table.

```
ANALYZE tablename;
```

- Update statistics on the entire database.

```
ANALYZE;
```

- Collect statistics on the **col_1** and **col_2** columns of the **tablename** table.

```
ANALYZE tablename (col_1, col_2);
```

- Add the declaration of statistics on the **col_1** and **col_2** columns in the **tablename** table.

```
ALTER TABLE tablename ADD STATISTICS (col_1, col_2);
```

Statistics Information View

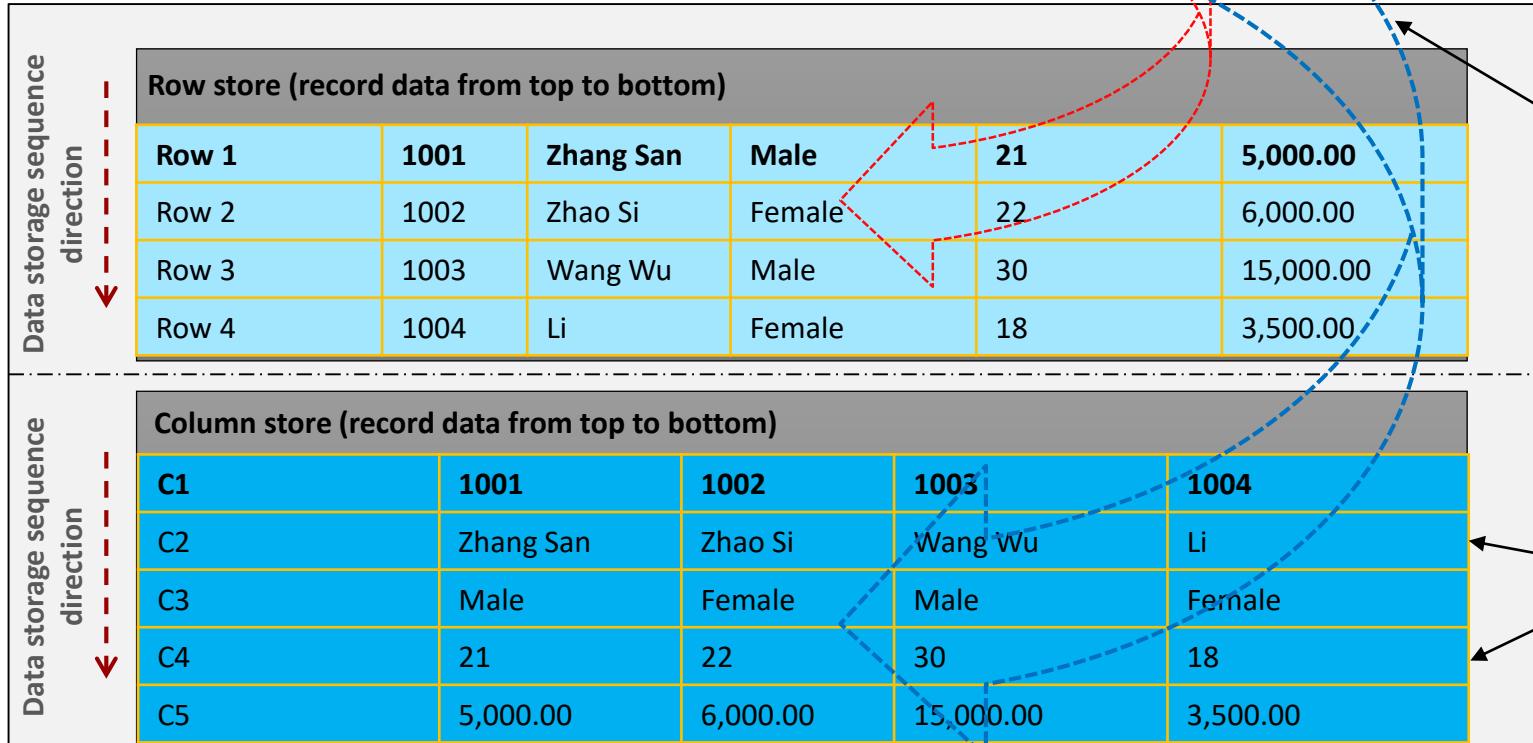
- **PG_STATISTIC** records database statistics on tables and index columns. This system catalog is accessible only to system administrators. Users can access this view only after being authorized.

Name	Type	Description
starelid	oid	Table or index to which the described column belongs.
starelkind	char	Object type.
staattnum	smallint	Number of the described column in the table, starting from 1.
stainherit	Boolean	Determines whether to collect statistics on objects that have inheritance relationships.
stanullfrac	real	Ratio of NULL records to all records in the column.
stawidth	integer	Average storage width of non-null records, in bytes.
stadistinct	real	Number of distinct, not-null data values in the column for all DNs.
stakindN	smallint	Code number stating that the type of statistics is stored in slot <i>N</i> of the pg_statistic row.
staopN	oid	Operator used to generate the statistics stored in slot <i>N</i> .
stanumbers	real[]	Column data values of the appropriate type for slot <i>N</i> . The value is NULL if the slot type does not store any data values.

Row Store and Column Store

Employee ID	Name	Gender	Age	Salary
1001	Zhang San	Male	21	5,000.00
1002	Zhao Si	Female	22	6,000.00
1003	Wang Wu	Male	30	15,000.00
1004	Li	Female	18	3,500.00

- There are two types of tables, each with their own way of storing data:
 - Row-store tables
 - Column-store tables



During the write process, records are split, and data in different columns is written to different storage areas. The extra write processes increase the I/O and reduce efficiency.

select name,age from employee;
For column-store table queries, only a small amount of storage corresponding to the required columns needs to be scanned, causing low I/O overhead.

Comparison Between Row Store and Column Store

- Row-store table
 - Data in the table is stored in rows, and all attributes in each row are stored together.
 - Row-store tables are used to query all attributes of a row.
 - INSERT and UPDATE operations are efficient.
 - Row-store table is the default storage mode.
- Column-store table
 - Table data is stored in columns, and multiple records in each column are stored together.
 - Column-store tables are used to query massive data and can reduce the volume of accessed disk data. The projection is efficient.
 - INSERT and UPDATE operations are slightly inefficient.

Hybrid Row-Column Store

- Use the keyword **ORIENTATION** to specify the storage mode.
 - ROW
 - COLUMN
- Use the keyword **COMPRESSION** to specify the compression level.
 - The higher the compression level is, the higher the compression ratio is.

Partitioning (1)

- In openGauss, data is partitioned horizontally on a node using a specified policy. This operation splits table data into multiple, non-overlapping partitions.
- In common scenarios, partitioned tables have the following advantages over common tables:
 - High query performance: Users can specify partitions when querying partitioned objects, improving query efficiency.
 - High availability: If a partition in a partitioned table is faulty, data in the other partitions is still available.
 - Easy maintenance: If a partition in a partitioned table is faulty, only the faulty partition needs to be repaired.
 - Balanced I/O: Partitions can be mapped to different disks to balance I/O and improve the overall system performance.

Partitioning (2)

- openGauss supports range partitioned tables, list partitioned tables, and hash partitioned tables.
 - 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 was created. This partitioning mode is most commonly used.
 - With range partitioning, 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 no overlap. If you specify the **PARTITION** parameter when running the **CREATE TABLE** statement, data in the table will be partitioned.
 - 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.
 - List partitioning 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.
 - Hash partitioned table: Data is mapped to each partition using the hash algorithm, and each partition stores records with the same hash value.
 - Hash partitioning uses an internal hash algorithm to divide records to be inserted into a table into partitions based on a column of the table.

Partitioning (3)

- 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) and obtain consecutive I/O to improve the query performance.
- In actual use, time is often used as a filter for query objects, so you can select the time column as the partition key. The key value range can be adjusted based on the total data volume and amount of data queried at a time.

Managing Partitioned Tables

Operation	Command
Create a partitioned table.	CREATE TABLE
Modify the row movement attributes of the partitioned table.	ALTER TABLE { ENABLE DISABLE } ROW MOVEMENT
Add a partition.	ALTER TABLE ADD PARTITION
Delete a partition.	ALTER TABLE DROP PARTITION
Rename a partition.	ALTER TABLE RENAME PARTITION
Truncate a partitioned table.	ALTER TABLE TRUNCATE PARTITION
Query a partition.	SELECT
Create an index for a partitioned table.	CREATE INDEX
Rename an index partition.	ALTER INDEX RENAME PARTITION

Basic Operations on Partitioned Tables (1)

- Creation example
 - Create a range partitioned table **student1**. The table has three partitions and their partition keys are integers: The value ranges of these three partitions are: **score1 < 60**, $60 \leq \text{score2} < 80$, and $80 \leq \text{score3} < 90$.

```
CREATE TABLE student1
(
    name varchar(30),
    score int,
    birthday timestamp
)
PARTITION BY RANGE(score)
(
    PARTITION score1 VALUES LESS THAN(60),
    PARTITION score2 VALUES LESS THAN(80),
    PARTITION score3 VALUES LESS THAN(90)
);
```

- Deletion example
 - Delete partition **score1**.

```
ALTER TABLE student1 DROP PARTITION score1;
```

Basic Operations on Partitioned Tables (2)

- Addition example
 - Add a partition **score4**, and the value range is $90 \leq \text{score4} \leq 100$.

```
ALTER TABLE student1 ADD PARTITION score4 VALUES LESS THAN (101);
```

- Renaming example
 - Rename partition **score2** as **score2_new**.

```
ALTER TABLE student1 RENAME PARTITION score2 TO score2_new;
```

- Query example
 - Query partition **score2**.

```
SELECT * FROM student1 PARTITION (score2_new);
```

Basic Operations on Partitioned Tables (3)

- Partitioned table index
 - Create a partitioned table index **student1_index1**, without specifying index partitions.

```
CREATE INDEX student1_index1 ON student1 (score) LOCAL;
```

- Create the partitioned table index **student1_index2** with the partition name specified.

```
CREATE INDEX student1_index2 ON student1 (score) LOCAL
(
    PARTITION score1_index,
    PARTITION score2_index TABLESPACE ts2,
    PARTITION score3_index TABLESPACE ts3
) TABLESPACE ts0;
```

- Rename an index partition.

```
ALTER INDEX student1_index2 RENAME PARTITION score1_index TO score1_index_new;
```

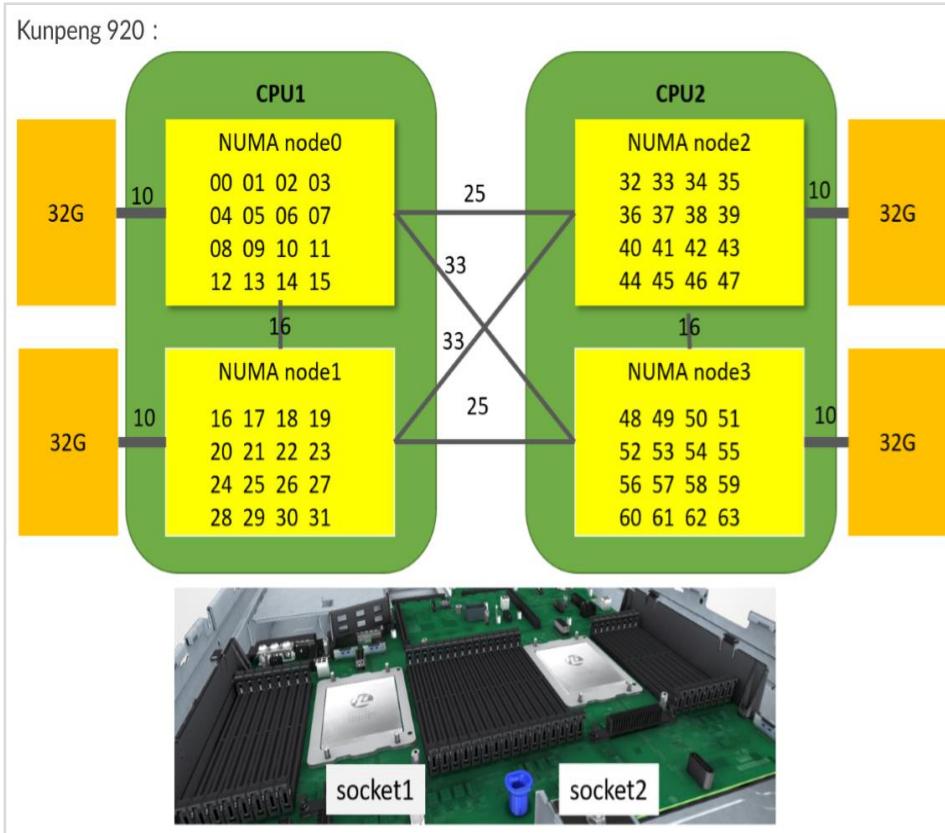
Partition Design Summary

- To partition the existing tables, you are advised to avoid creating indexes before data migration is complete.
- To take advantage of the query characteristics of partitioned tables, use the fields used during partitioning as the filter criteria.
- After partitions are created, the UUIDs of the partitions can be the same.
- Querying data using partition keys is efficient.
- The partition columns cannot be empty. For example, the case filing date or case closing date cannot be used as the partition column.
- You can back up each partition separately. However, if you want to back up all of the partitions, you can back up only the entire schema.

Kunpeng NUMA Architecture Optimization

- openGauss uses a multi-core Kunpeng NUMA architecture that has been optimized to reduce cross-core memory access latency and maximize multi-core Kunpeng computing capabilities. The key technologies include redo log batch insertion, NUMA distribution of hotspot data, and Clog partitions, greatly improving the processing performance of the TP system.
- openGauss uses an LSE instruction set based on ARMv8.1 for efficient atomic operations, effectively improving the CPU usage, multi-thread synchronization performance, and Xlog write performance.
- openGauss uses the wider L3 cache line provided by the Kunpeng chip to improve 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 NUMA CPU Structure



- Non-uniform memory access (NUMA)
- NUMA distance: the physical distance between the processor and memory block of a NUMA node. You can use the numactl tool to query the CPU access distance.

Node/Node distances	0	1	2	3
0	10	16	32	33
1	16	10	25	32
2	32	25	10	16
3	33	32	16	10

NUMA ARM Atomic Instruction

- The Compare and Swap (CAS) operation is used to ensure the consistency of data written during a multi-thread operation.
- Uninterrupted data exchange operations are used for multi-thread programming to avoid data inconsistencies caused by execution sequence uncertainty and interrupt unpredictability when a piece of data is while an operation is in progress.

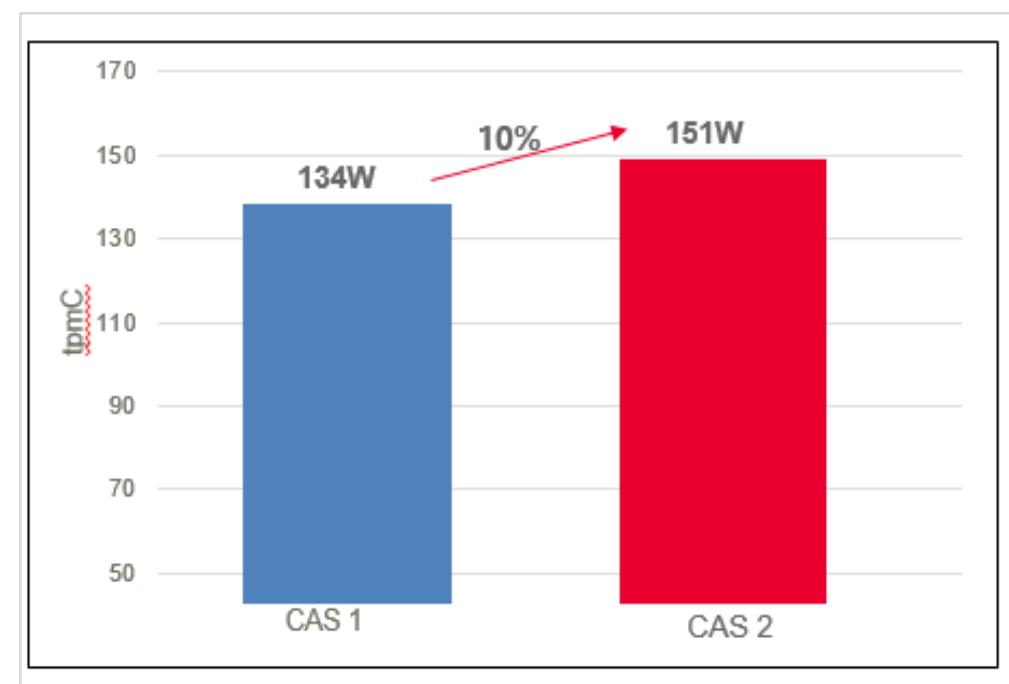
CAS implementation using ldxr and stlxr instructions

```
20     os_compare_and_swap_ulong(&old,old,new);
0x000000000004005bc <+24>:    ldr    w0, [x29,#24]
0x000000000004005c0 <+28>:    mov    w2, w0
0x000000000004005c4 <+32>:    ldr    w1, [x29,#28]
0x000000000004005c8 <+36>:    add    x0, x29, #0x18
0x000000000004005cc <+40>:    ldxr   w3, [x0]
0x000000000004005d0 <+44>:    cmp    w3, w2
0x000000000004005d4 <+48>:    b.ne   0x4005e0 <main+60>
0x000000000004005d8 <+52>:    stlxr  w4, w1, [x0]
0x000000000004005dc <+56>:    cbnz   w4, 0x4005cc <main+40>
0x000000000004005e0 <+60>:    dmb    ish
```

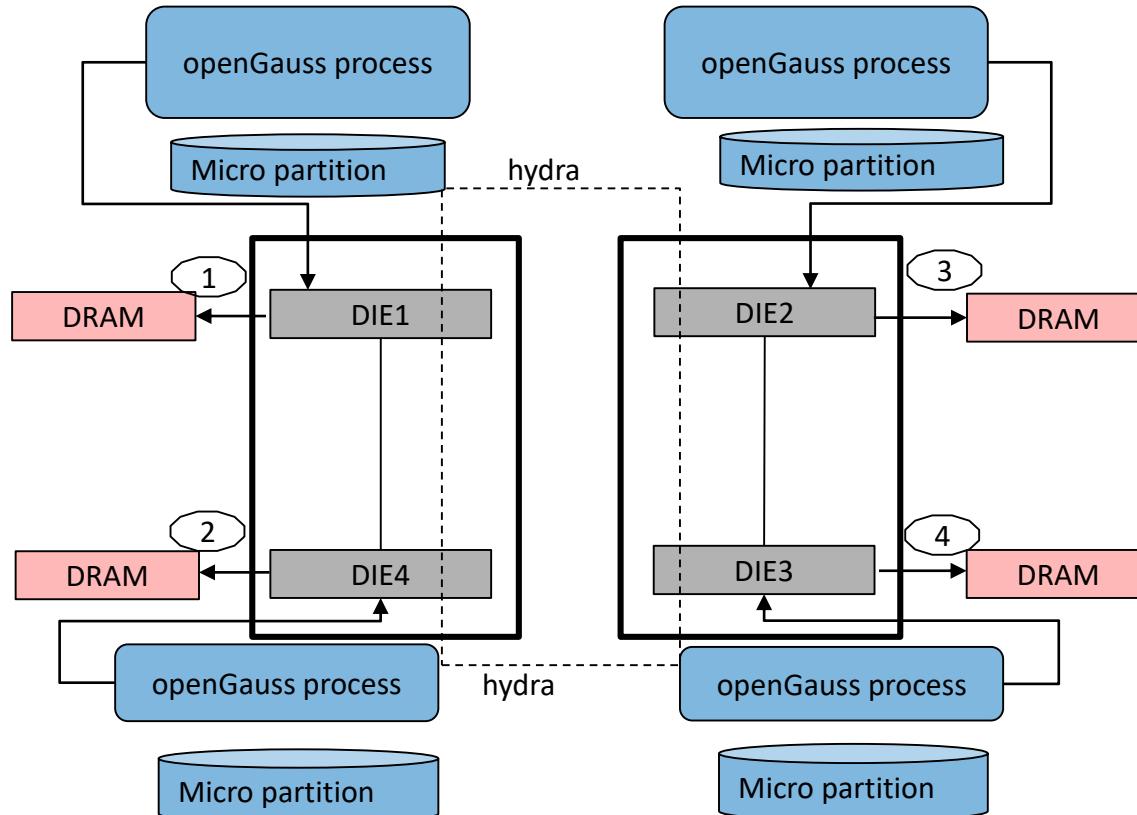
CAS implementation using casal instruction

```
4005b8:      b9001ba0      stl    w0, [x29,#24]
// int *ptr=100;
os_compare_and_swap_ulong(&old,old,new);
4005bc:      b9401ba0      ldr    w0, [x29,#24]
4005c0:      2a0003e2      mov    w2, w0
4005c4:      b9401fa1      ldr    w1, [x29,#28]
4005c8:      910063a0      add    x0, x29, #0x18
4005cc:      2a0203e3      mov    w3, w2
4005d0:      88e3fc01      casal  w3, w1, [x0]
4005d4:      6b02007f      cmp    w3, w2
```

Four instructions → one instruction = more efficient execution

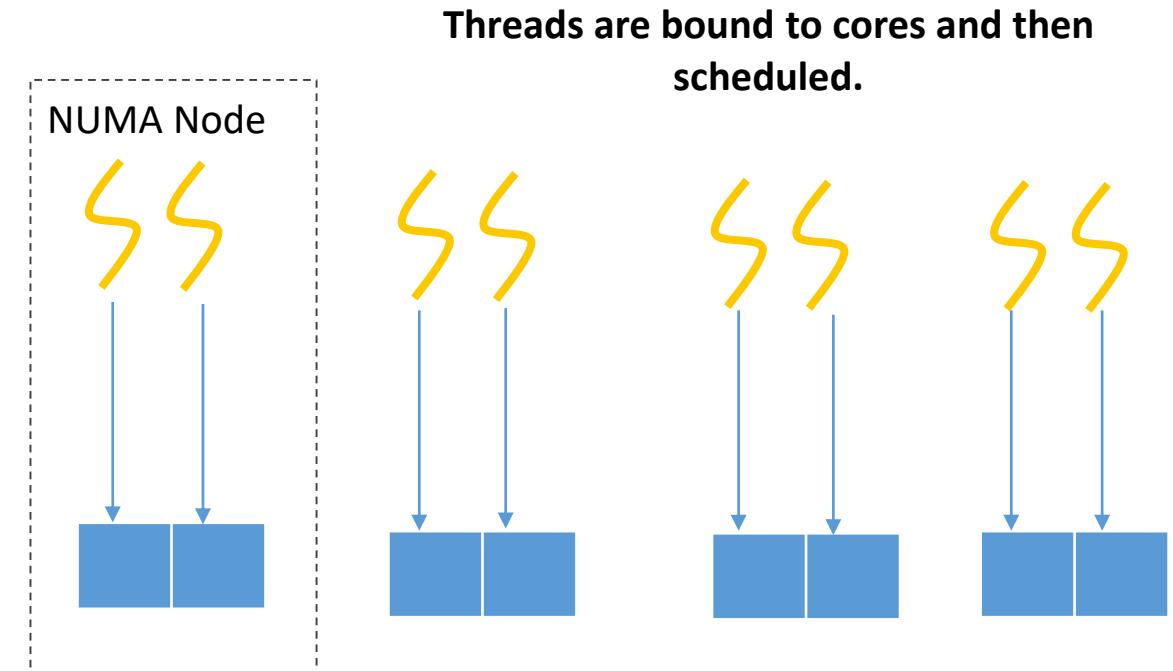
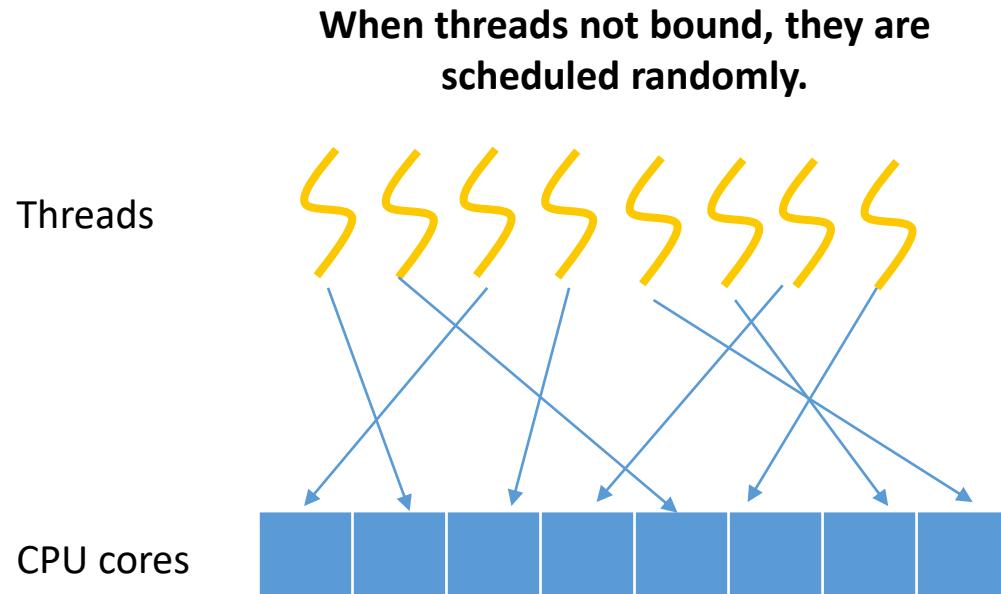


NUMA Kernel Data Structure



- Threads are bound to cores to prevent thread offsets between cores.
- The NUMA-based data structure has been reconstructed to reduce cross-core access.
- Data is partitioned to reduce thread access conflicts.
- The algorithm has been adjusted to reduce single-point bottlenecks.
- ARM atomic instructions are used to reduce the computing overhead.

Enabling NUMA for Binding Threads to Cores



- Network interruption and background service threads are bound to cores separately.
- Different service threads are bound to different cores.

NUMA Multi-core Optimization Results

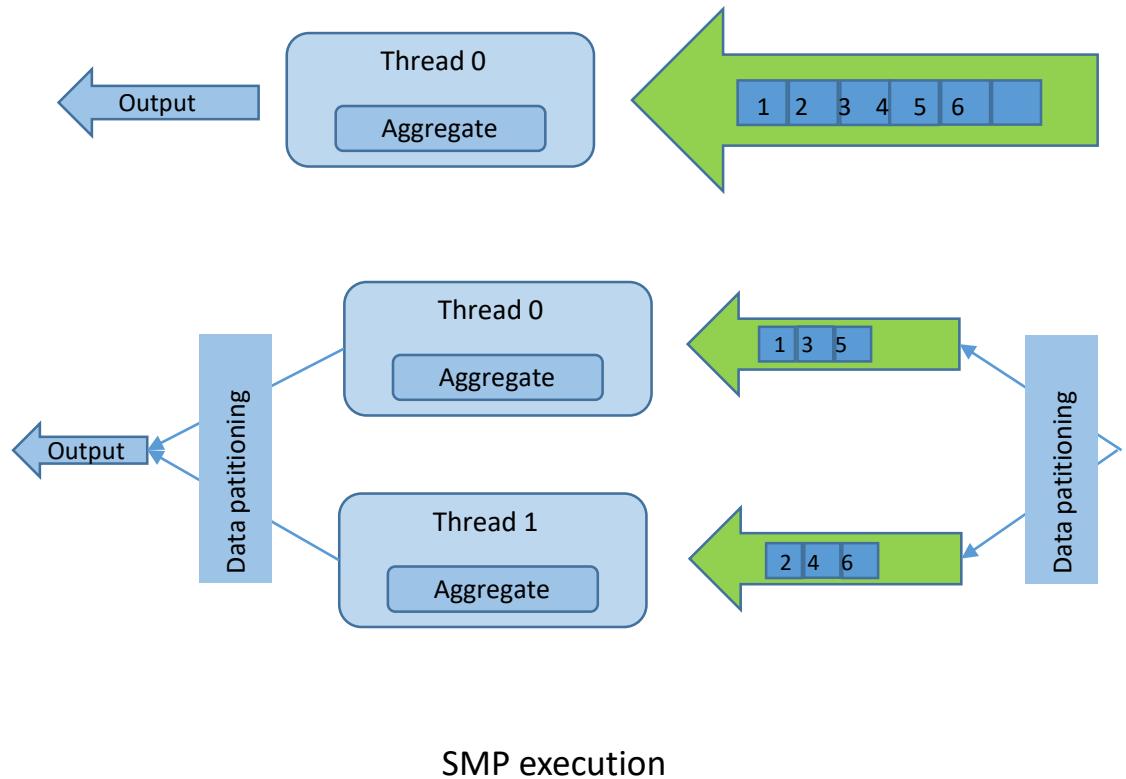
The terminal window displays CPU usage statistics for four cores (3, 5, 7, 8) over time. The output shows various processes running, with CPU utilization percentages ranging from 94.2% to 96.1%. The system load average is 357.50, and the uptime is 1 day, 02:16:36.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
28118	o-mm	20	0	8446	153G	139G	S	10437	30.2	23h23:47	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
29073	o-mm	20	0	8446	153G	139G	R	54.9	30.2	7:57:30	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
29074	o-mm	20	0	8446	153G	139G	S	25.2	30.2	3:28:32	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28970	o-mm	20	0	8446	153G	139G	S	13.6	30.2	1:42:57	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28627	o-mm	20	0	8446	153G	139G	S	13.6	30.2	1:41:40	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28905	o-mm	20	0	8446	153G	139G	R	13.6	30.2	1:43:88	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28749	o-mm	20	0	8446	153G	139G	S	13.6	30.2	1:42:14	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28773	o-mm	20	0	8446	153G	139G	R	11.0	30.2	1:42:25	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28534	o-mm	20	0	8446	153G	139G	S	11.6	30.2	1:41:34	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28769	o-mm	20	0	8446	153G	139G	S	13.1	30.2	1:42:32	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
29254	o-mm	20	0	8446	153G	139G	S	21.3	30.2	1:07:12	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28853	o-mm	20	0	8446	153G	139G	S	12.3	30.2	1:43:17	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28949	o-mm	20	0	8446	153G	139G	S	13.6	30.2	1:43:40	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28959	o-mm	20	0	8446	153G	139G	R	11.6	30.2	1:42:59	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28940	o-mm	20	0	8446	153G	139G	R	12.9	30.2	1:41:38	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28542	o-mm	20	0	8446	153G	139G	R	11.6	30.2	1:41:85	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28780	o-mm	20	0	8446	153G	139G	R	12.3	30.2	1:42:31	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn
28858	o-mm	20	0	8446	153G	139G	R	12.3	30.2	1:42:56	/data/huawei/install/app/bin/gaussdb --single_node -D /data/huawei/install/data/dn

The TPC-C test performance reaches 1,500,000 tpmC based on Kunpeng 920.
The CPU efficiency is close to 95%.

Server	Project	Model	Quantity
	CPU	Kunpeng 920/64 cores/64 threads	2 (Each CPU contains 64 cores, and there are 128 cores in total.)
Taishan 2280 V2	Frequency	2.6 GHz	---
	Memory	DDR4, 2933MT/s	512 GB
	Network	Hi1822	
	Drive	NVMe; 3.2T	4
	OS	openEuler-20.03-LTS	1

SMP Technology (1)



- openGauss symmetric Multi-Processing (SMP) uses the multi-core CPU architecture of a computer for multi-thread parallel computing, taking full advantage of CPU resources to improve query performance.
- SMP parallel execution refers to the parallel execution of tasks at the **thread** level. Theoretically, there can be as many subtasks executed concurrently as there are cores on the physical server.
- SMP parallel threads are part of the same process, and **data can be exchanged directly through the memory**, without occupying network connections or bandwidth. This reduces the impact of network factors that restrict the performance improvement of the MPP system.
- Finally, because the parallel subtasks **do not need to be attached to another background worker thread** after being started, the utilization rate of system computing resources can be increased.

SMP Technology (2)

- The database requires not only Massively Parallel Processing (MPP) for good horizontal scale-out, but also SMP for better scale-up on a single node.
- SMP can improve the database performance in two ways:
 - It significantly reduces the time needed to execute a single query.
 - It improves the system throughput for a given period of time and improves system resource utilization.
- It uses multi-thread and multi-subtask parallel execution to efficiently maximize the use of system computing resources. Lightweight SMP multi-thread execution can solve problems involved with MPP architecture deployment.

Contents

1. System Architectures
2. Deployment Solutions
3. Typical Networking
- 4. Key Features**

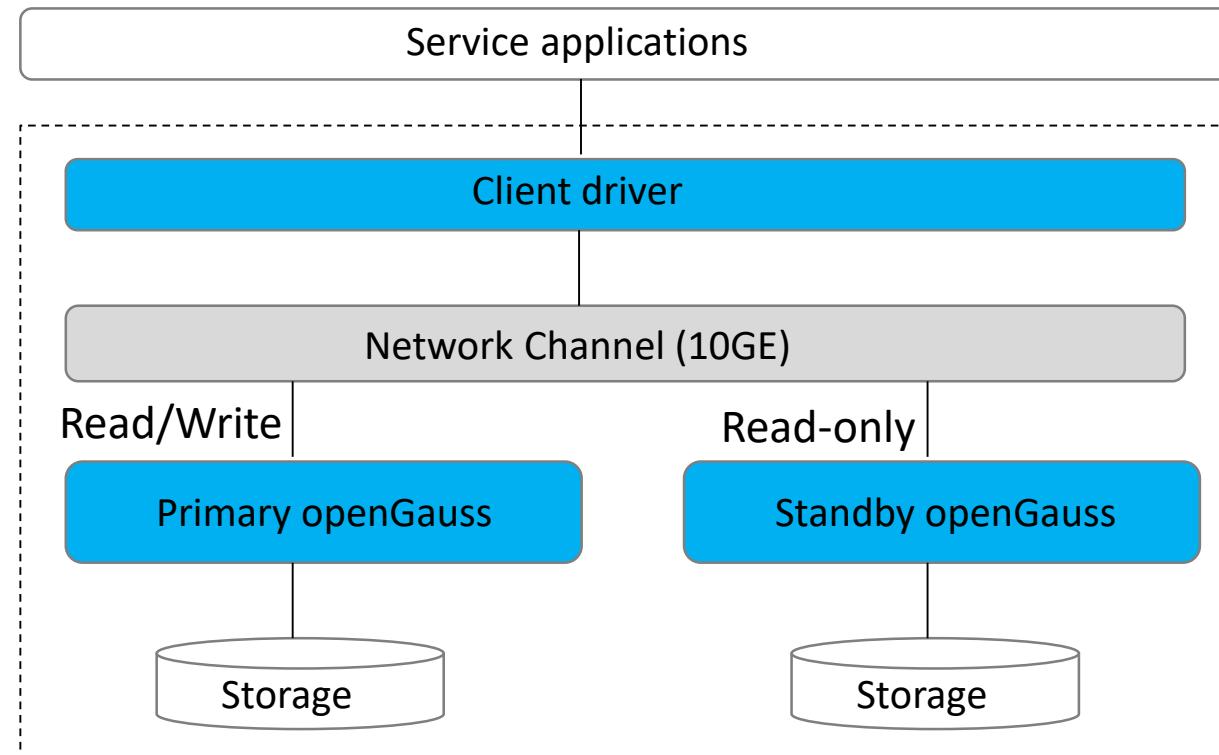
- High Performance
- **High Availability (HA)**
- High Security
- Easy Maintenance
- AI Capabilities

High Availability Architecture

- To ensure that a fault can be rectified, there needs to be multiple copies of the data available. Multiple copies are configured for the primary and standby nodes, and logs are used for data synchronization. In this way, with openGauss, no data is lost if a node fails or if the system restarts after a stop. openGauss meets ACID requirements.
- There are two types of primary/standby deployment modes:
 - Primary/Standby
 - One primary and multiple standbys
- The **switchover** command can be used to trigger a switchover between the primary and standby nodes. If the primary node fails, the **failover** command can be used to promote the standby node to primary.

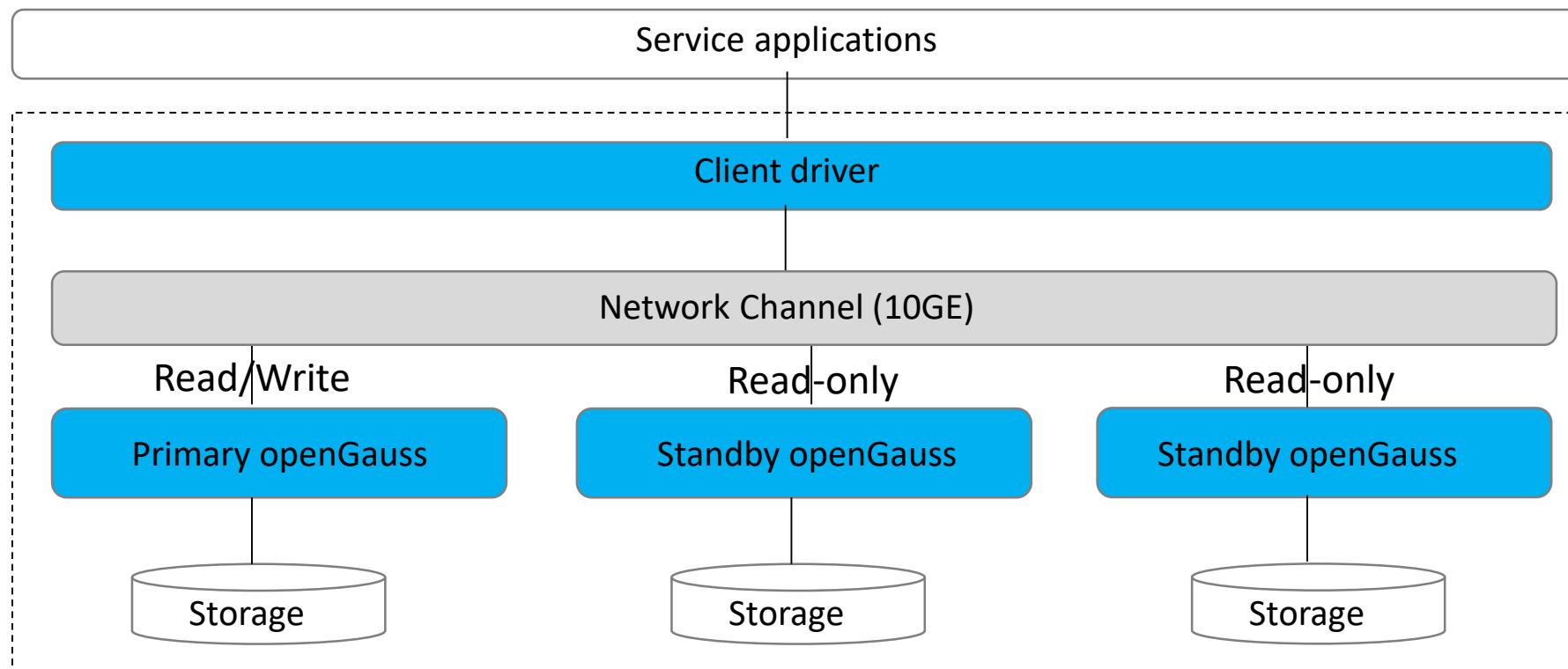
Primary/Standby Architecture

- In a primary/standby deployment, there are two data copies, one for the primary node and another for the standby node. The standby node receives and plays back the logs.



One Primary and Multiple Standbys Architecture

- The one primary and multiple standbys mode provides more robust DR and is suitable for OLTP systems with higher availability transaction processing requirements.



Performing a Switchover (1)

- Scenario
 - The database administrator may need to manually perform an primary/standby switchover on an openGauss database node that is currently running. For example, after a primary/standby database node failover, you may need to restore the original primary/standby roles or to manually perform a primary/standby switchover due to a hardware fault. A cascaded standby server cannot be directly upgraded to primary. You need to first perform a switchover or failover to change the cascaded standby server to a standby server, and then to a primary server.
- Procedure
 - Log in to any database node as the OS user **omm** and run the following command to check the primary/standby status:
 - `gs_om -t status --detail`
 - `gs_ctl switchover -D /home/omm/cluster/dn1/`

Performing a Switchover (2)

- If the primary node is faulty, run the following command on the standby node:

```
gs_ctl failover -D /home/omm/cluster/dn1/
```

- After the switchover or failover is successful, run the following command to record information about the current primary and standby nodes:

```
gs_om -t refreshconf
```

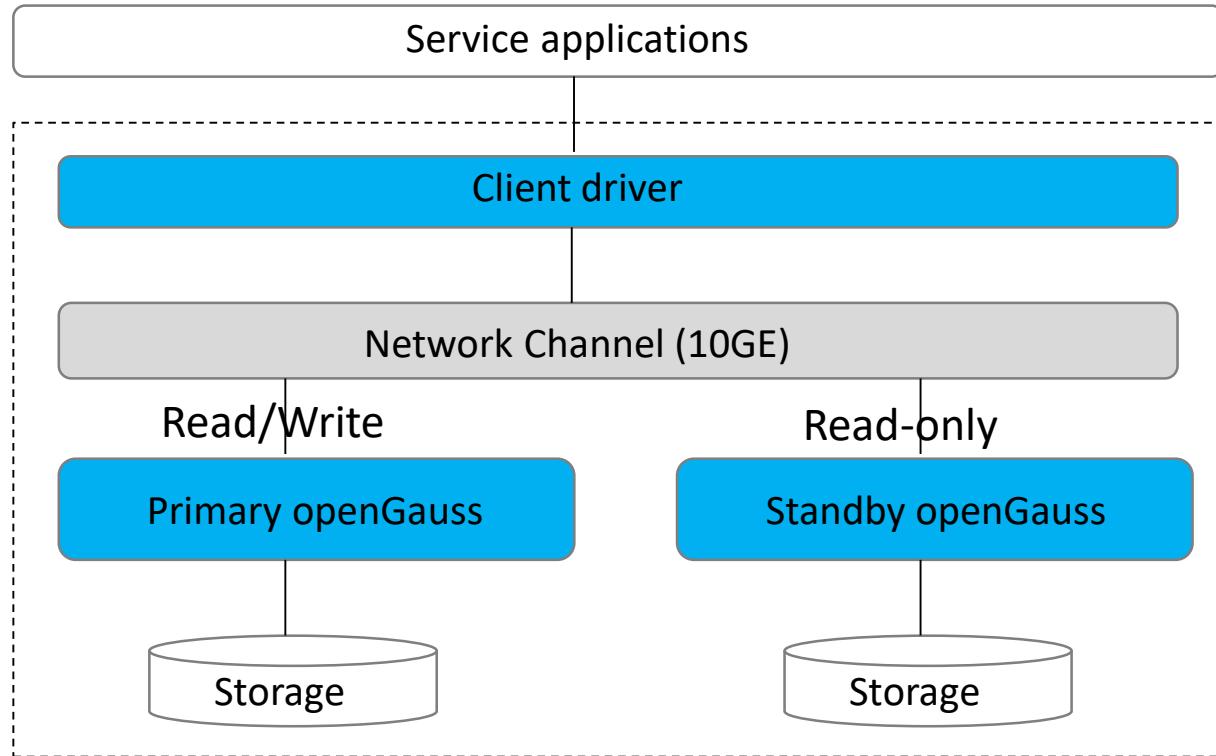
High Availability - Primary and Standby Nodes

- Multi-copy deployment protects against instance-level faults. This mode is good for scenarios where equipment room DR is not required but some hardware faults need to be protected against.
- Generally, multi-copy deployment means one primary and two standbys (three copies total). Three copies give you 99.99% reliability, which is enough for most applications.
 - In primary/standby quorum replication, data can be synchronized to multiple nodes but using only one standby node ensure the best performance.
 - If any of the primary or standby nodes fails, services will not be affected.
 - There are three copies of the data, so even if one node fails, the system still has two copies of the data available. In addition, any standby DN can be promoted to primary.
 - The primary and standby nodes cannot be deployed on the same physical machine.

Optimal RTO (1)

- Concepts
- RTO
 - Recovery Time Objective
 - Application-oriented
 - It is the amount of application downtime that can be tolerated.
- RPO
 - Recovery Point Objective
 - Data-oriented
 - It is the amount of lost data that can be tolerated.

Optimal RTO (2)



- Recovery Time Objective (RTO) is the maximum amount of acceptable downtime for the entire system after a disaster occurs.

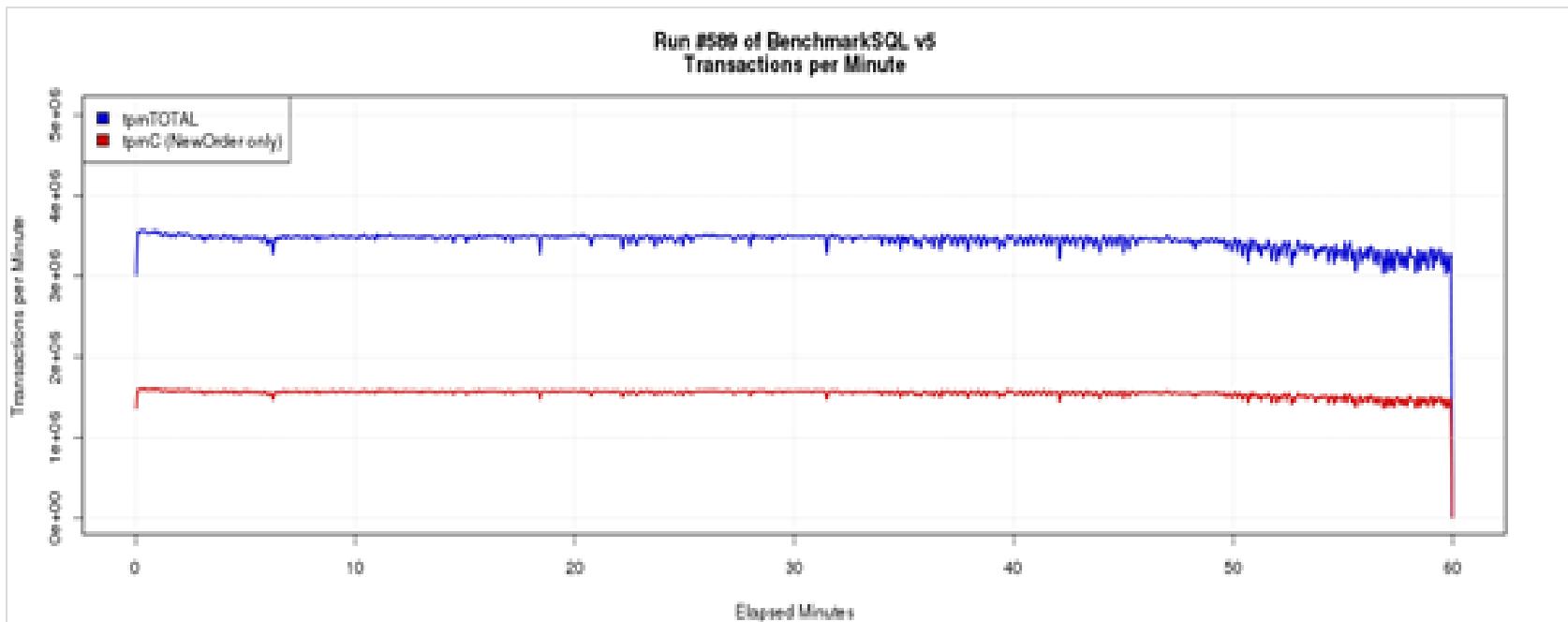
The duration from the time when a service is interrupted to the time when the service is restored. This indicator reflects the timeliness of service recovery.

- Recovery Point Objective (RPO) is the longest acceptable period for which data might be lost.

Amount of lost data after the service system is recovered. This indicator reflects the data integrity.

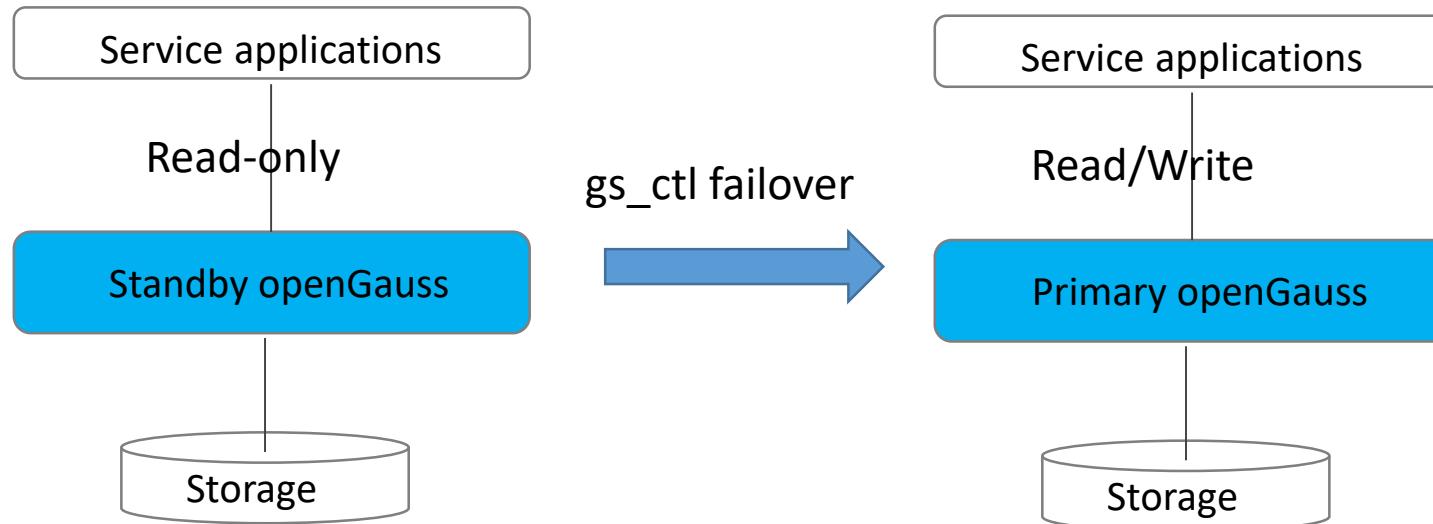
Optimal RTO (3)

- Incremental checkpoints more efficient, more frequent checkpoints; less downtime, and a lower RTO.



openGauss RTO Time

With a 60% load and over 700,000 tpmC, the RTO is less than 10 seconds. (The standby node takes over services within 10 seconds of when the switchover command is executed.)



Contents

1. System Architectures
2. Deployment Solutions
3. Typical Networking
- 4. Key Features**

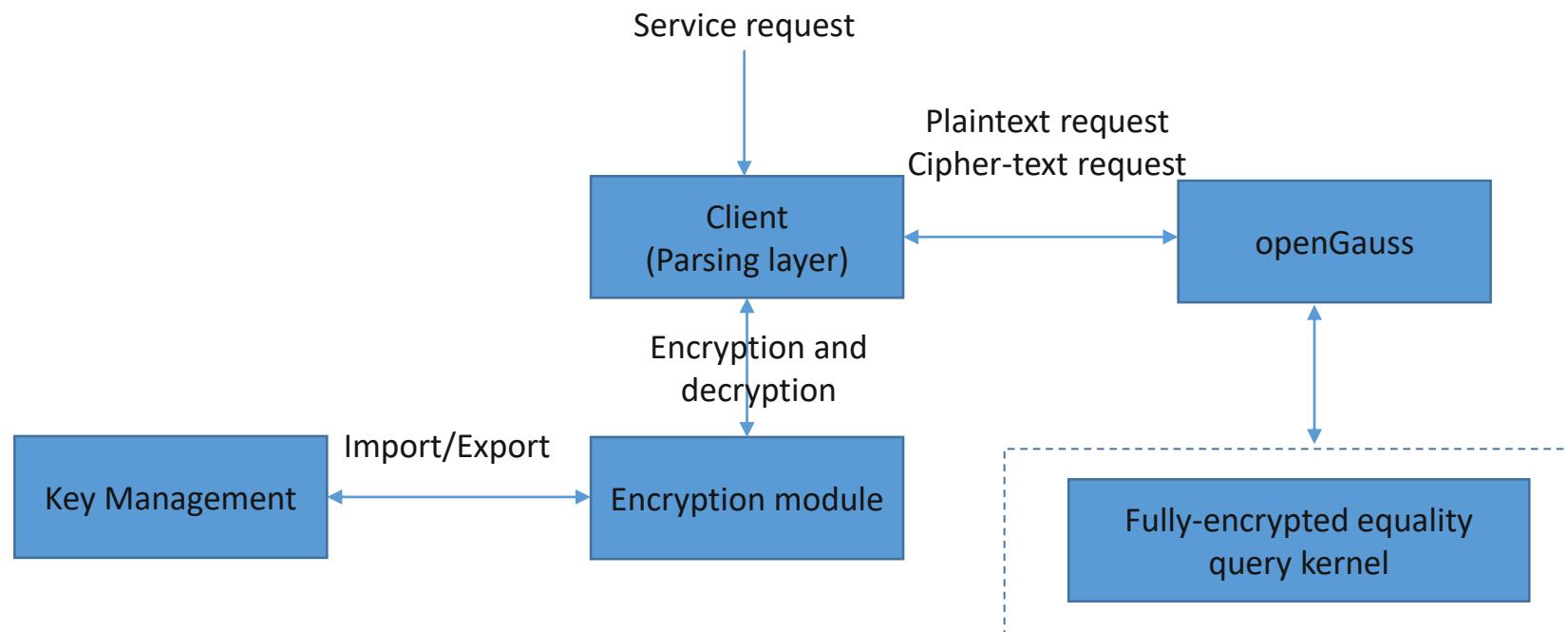
- High Performance
- High Availability (HA)
- **High Security**
- Easy Maintenance
- AI Capabilities

Equality Query in a Fully-encrypted Database

- An encrypted database aims to protect privacy throughout the data lifecycle. Data is always encrypted during transmission, computing, and storage regardless of the service scenario or environment.
- After the data owner encrypts data on the client and sends the encrypted data to the server, even if an attacker manages to exploit some system vulnerability and steal user data, they cannot obtain valuable information. Data privacy is protected.
- The entire service data flow is encrypted during processing. A fully-encrypted database:
 - Protects data privacy and security throughout the lifecycle on the cloud. Attackers cannot obtain information from the database server regardless of the data status.
 - Helps cloud service providers earn the trust of third-party users. Users, including service administrators and O&M administrators in enterprise service scenarios and application developers in consumer cloud services, can keep the encryption keys themselves so that even users with high permissions cannot access unencrypted data.
 - Enables cloud databases to better comply with personal privacy protection laws and regulations.

Fully-encrypted Equality Query Process

- Three-layer key management: root key, master key, and column encryption key.
- The client encrypts and decrypts data, and the server processes the encrypted data.
- Columns that do not need to be encrypted are still processed in plaintext.



Equality Query in a Fully-encrypted Database (1)

- Encrypted equality query supports the following data types:

Data	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 p .
	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.

Equality Query in a Fully-encrypted Database (2)

Data	Type	Description
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 n 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.

Dynamic Data Masking (1)

- Data privacy is a required database security capability. Unauthorized access to user data need to be restricted and data security needs to be ensured.
- Dynamic data masking protects data privacy with customizable masking policies. It can prevent unauthorized users from accessing sensitive information without modifying existing data.
- When the administrator specifies an object to be anonymized and creates a custom 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.

Dynamic Data Masking (2)

- Dynamic data masking includes custom masking policies based on resource labels. You can select masking modes based on the site requirements or create custom masking policies for specific users. The SQL syntax for creating a 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);
```

- **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.

Dynamic Data Masking (3)

- The following table shows the preconfigured masking functions:

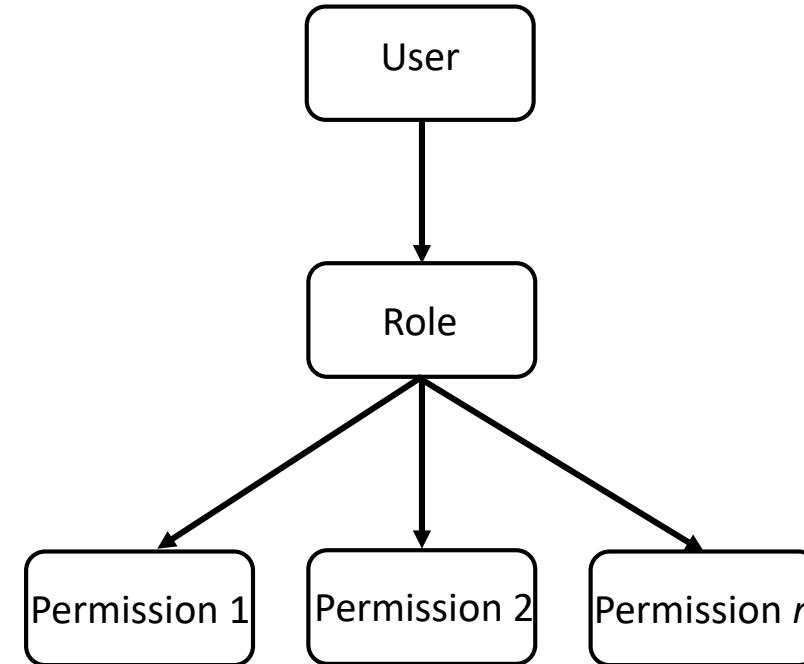
Masking Function	Example
creditcardmasking	'4880-9898-4545-2525' will be displayed as 'xxxx-xxxx-xxxx-2525'. This function masks digits except the last four digits.
basicemailmasking	'abcd@mymail.com' will be displayed as 'xxxx@mymail.com'. This function masks all text before the @.
fullemailmasking	'abcd@mymail.com' will be displayed as 'xxxx@xxxxx.com'. This function masks all text other than the @ before the first dot (.)
alldigitsmasking	'alex123alex' will be displayed as 'alex000alex'. This function masks only the digits.
shufflemasking	'hello world' will be displayed as something like 'hlwoeor dl'. This weak masking function is implemented through character dislocation. You are not advised to use this function to mask strings with strong semantics.
randommasking	'hello world' will be displayed as something like 'ad5f5ghdf5'. This function randomly masks text by character.
maskall	'4880-9898-4545-2525' will be displayed as 'xxxxxxxxxxxxxxxxxx'.

Access Control

- Access control is the management of user access to a database through permissions management, including database system and object permissions.
- Role-based access control is supported. Different permissions sets are associated with different roles. Permissions are assigned to roles and then roles are assigned to users. The login access control is implemented by using the user ID and authentication technology. Object access control is when access is controlled by configuring user permissions for different objects. To improve security, you can assign database users the minimum permissions required for completing different tasks.
- Access control based separation of duties is supported. Database roles include system administrator, security administrator, and audit administrator. The security administrator creates and manages users, the system administrator grants and revokes user permissions, and the audit administrator audits all user actions.
- Role-based access control model is used by default, but you can also use parameter various parameters to configure separation of duties for your users.

Role-Based Access Control (1)

- What is role-based access control (RBAC)?
 - Permissions are assigned to roles, and then the users associated with those roles inherit the permissions.
 - RBAC greatly simplifies permissions management.
- What is the RBAC model?
 - Assign appropriate permissions to roles.
 - Associate users with the roles.



Role-Based Access Control (2)

- RBAC authorization defines relationships between who, what, and how.
 - Who: the owner or subject of the permissions (for example, a user).
 - What: the object the permissions apply to (such as tables and functions).
 - How: specific permissions (positive or negative authorization).
- Relationships between users, roles, and permissions in the RBAC model are as follows:
 - A user can have multiple roles.
 - A role can have multiple users assigned.
 - A role can have multiple permissions associated.
 - A permission can be assigned to multiple roles.

Role-Based Access Control (3)

- Other access control models
 - Access control lists (ACL)
 - Attribute-Based access control (ABAC)
 - Policy-Based access control (PBAC)
- RBAC features and advantages
 - Indirect relationships
 - Responsibility separation
 - Efficient permissions management
 - Principle of least privilege, differentiated responsibilities, and data abstraction

Row-Level Security (1)

- Row-level security is database access control at the table row level.
- When different users perform the same SQL query operation, the read results may be different.
- 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 RLS policy 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 security is used to control the visibility of row-level table data. 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**.

Row-Level Security (2)

- Enable row-level security.

```
ALTER TABLE tablename ENABLE ROW LEVEL SECURITY;
```

- Create a row-level security policy to specify that the current user can view only their own data.

```
CREATE ROW LEVEL SECURITY POLICY tablename_rls ON tablename USING(role = CURRENT_USER);
```

— Note: *tablename* indicates the name of the created table, and *tablename_rls* indicates the name of the created row-level security policy.

Database Audit

- The audit logs record user operations performed on database startup and stopping, connection, and DDL, DML, and DCL operations. The audit log enhances the database's ability to trace illegal operations.
- You can set parameters to specify the statements or operations are logged.
- Audit logs record the event time, type, execution result, username, database, connection information, database object, database instance name, port number, and other details. You can query audit logs by start time and end time and filter audit logs by recorded column.
- Database security administrators can use audit logs to reproduce a series of events that caused a fault in the database or to identify unauthorized users, unauthorized operations, and the time when these operations were performed.

Unified Auditing

- In addition to the preceding data auditing functions, there is also support for unified auditing. Security audits can be made more efficient through the use of custom audit policies. The administrator defines audit objects and behaviors. If a task associated with an audit policy is executed, the corresponding audit behavior is generated and a log entry is recorded. Custom audit policies can cover common user management activities, as well as DDL and DML operations, meeting routine audit requirements.

Unified Auditing

- Administrators can configure audit policies for database resources or resource labels to simplify management, generate audit logs, reduce redundant audit logs, and improve management efficiency.
- Administrators can create custom policies for configuring operation behaviors or database resources. The policies are used to audit specific user scenarios, user behaviors, or database resources. If centralized auditing is enabled when a user accesses the database, the system matches the corresponding audit policy based on user details, such as the access IP address, client tool, and username. Then, the system classifies the user behaviors based on the access resource labels and user operation types (DML or DDL) in the policy.
- The purpose of unified auditing is to include specific behaviors in an audit and exclude others, thereby simplifying management and improving the security of audit data generated by the database.

Creating an Auditing Policy

- Only the poladmin, sysadmin, or initial user can create an auditing policy. In addition, you need to enable the security policy, that is, set the GUC parameter **enable_security_policy** to **on** to make the policy take effect.

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { { privilege_audit_clause | access_audit_clause }  
[ filter_group_clause ] [ ENABLE | DISABLE ] };  
• privilege_audit_clause:  
PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ...] ) ]  
• access_audit_clause:  
ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ...] ) ]  
• filter_group_clause:  
FILTER ON { ( FILTER_TYPE ( filter_value [, ...] ) ) [, ...] }
```

Example of Creating an Auditing Policy (1)

- Step 1: Create the **dev_audit** and **bob_audit** users.

```
CREATE USER dev_audit PASSWORD 'dev@1234';
CREATE USER bob_audit password 'bob@1234';
```

- Step 2: Create the **tb_for_audit** table.

```
CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);
```

- Step 3: Create a resource label.

```
CREATE RESOURCE LABEL adt_lb0 add TABLE(tb_for_audit);
```

- Step 4: Perform the **CREATE** operation on the database to create an auditing policy.

```
CREATE AUDIT POLICY adt2 ACCESS SELECT;
```

Example of Creating an Auditing Policy (2)

- Step 5: Create an audit policy to audit only the **CREATE** operations performed on the **adt_lb0** resource by users **dev_audit** and **bob_audit**.

```
CREATE AUDIT POLICY adt3 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON  
ROLES(dev_audit, bob_audit);
```

- Create an audit policy to audit only the **SELECT**, **INSERT**, and **DELETE** operations performed on the **adt_lb0** resource by users **dev_audit** and **bob_audit** using client tools **psql** and **gsql** on the servers whose IP addresses are **10.20.30.40** and **127.0.0.0/24**.

```
CREATE AUDIT POLICY adt4 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON LABEL(adt_lb0), DELETE FILTER ON  
ROLES(dev_audit, bob_audit), APP(pssql, gsql), IP('10.20.30.40', '127.0.0.0/24');
```

Modifying or Deleting an Auditing Policy

- Only the poladmin, sysadmin, or initial user can modify or delete the unified auditing policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { [ privilege_audit_clause ]  
[ access_audit_clause ] };
```

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
```

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
```

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```

System Catalogs Related to Unified Auditing

- **GS_AUDITING_POLICY** records the main information about the audit. Each record corresponds to a design policy.
- **GS_AUDITING_POLICY_ACCESS** records the audit information for DML database operations.
- **GS_AUDITING_POLICY_FILTERS** records the filtering policies of the audit. Each record corresponds to a design policy.
- **GS_AUDITING_POLICY_PRIVILEGES** records the DDL database operations for the audit. Each record corresponds to a design policy.

Contents

1. System Architectures
2. Deployment Solutions
3. Typical Networking

4. Key Features

- High Performance
- High Availability (HA)
- High Security
- Easy Maintenance
- AI Capabilities

Workload Diagnosis Report (WDR) Overview

- The workload diagnosis report (WDR) generates a performance report for the period of time between two snapshots. The report is used to diagnose database kernel performance faults.
- WDR is the main method for diagnosing long-term performance problems. Multiple aspects of system performance are analyzed and compared to snapshot baselines. These analyses inform DBAs of the system load and the performance of each component, so they can identify performance bottlenecks.
- Snapshots are also an important data source for subsequent performance problem self-diagnosis and self-optimization suggestions.

WDR

- 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 points in time, and generates a details summary of the performance data.

WDR Generation

- Prerequisites
 - A report can be generated after WDR snapshots are enabled (that is, **enable_wdr_snapshot** is set to **on**) and the number of snapshots is greater than or equal to 2.
- Procedure
 - Run the following command to query the generated snapshot and obtain **snapshot_id**:

```
select * from snapshot.snapshot;
```
 - (Optional) Run the following command to manually create a snapshot. If only one snapshot exists in the database or you want to view the monitoring data of the database in the current period, manually create a snapshot. This command is only available to the **sysadmin** user.

```
select create_wdr_snapshot();
```

Example of WDR Generation

- Query the snapshots that have been generated.

```
postgres=# select * from snapshot.snapshot;
snapshot_id |      start_ts      |      end_ts
-----+-----+-----+
 1 | 2020-09-07 10:20:36.763244+08 | 2020-09-07 10:20:42.166511+08
 2 | 2020-09-07 10:21:13.416352+08 | 2020-09-07 10:21:19.470911+08
```

- Write the queried information to a performance report:

```
select generate_wdr_report(begin_snap_id Oid, end_snap_id Oid, varchar report_type, varchar report_scope, int node_name );
```

- Generate the formatted performance report **wdrTestNode.html**.

```
postgres=# \a \t \o /home/om/wdrTestNode.html
Output format is unaligned.
Showing only tuples.
```

- Disable performance report generation.

```
postgres=# \o \a \t
Output format is aligned.
Tuples only is off.
```

Diagnosis Summaries

Diagnosis Type	Description
Database Stat	Evaluates the load and I/O status of the current database. Load and I/O are the most important characteristics of a TP system. 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.
Load Profile	Evaluates current the time, I/O, transaction, and SQL system loads. The statistics include the job running elapse time, CPU time, daily transaction volume, logical and physical read volume, I/O volume, login and logout times, SQL, transaction execution volume, and SQL P85 and P90 response time.
Instance Efficiency Percentages	Evaluates the cache efficiency of the current system. The statistics include the database cache hit ratio.
Events	Evaluates the performance of key system kernel resources and key events. The statistics include the number of times that the key events of the database kernel have occurred and the wait times.
Wait Classes	Evaluates the performance of key events in the system. The statistics include the release of the data kernel in the main types of waiting events, such as STATUS, LWLOCK_EVENT, LOCK_EVENT, and IO_EVENT.
CPU	The statistics include time release of the CPU in user, kernel, wait I/O, or idle modes.
IO Profile	The statistics include the database I/O count and volume, and the number and volume of I/O redos.
Memory Statistics	The statistics include maximum process memory, used process memory, maximum shared memory, and used shared memory.

Diagnosis Report Details

Diagnosis Type	Description
Time Model	Evaluates the performance of the current system based on how much time various items take. 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 a range of different normalized SQL performance indicators: 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 performance of backend jobs. The statistics include the performance of backend operations such as page operation and replication.
Object stats	Diagnoses the performance of database objects. The statistics include user tables, tables on indexes, index scan activities, insert, update, and delete activities; the number of valid rows, and table maintenance statuses.
Configuration settings	Determines whether a configuration has changed. The statistics include snapshots of all current configuration parameters.

Information Collection for One-Click Diagnosis (1)

- Multiple suites are provided to capture, collect, and analyze diagnosis data, enabling fault diagnosis and accelerating the diagnostic process. Necessary database logs, cluster management logs, and stack information can be extracted from the production environment based on the requirements of development and fault locating personnel. Fault locating personnel demarcate and locate faults based on this information.
- The one-click collection tool obtains different information from the production environment depending on the actual faults, improving the fault locating and demarcation efficiency. You can modify the configuration file based on the information you need to collect. For example, you can collect:
 - OS information by running Linux commands
 - Database information by querying system catalogs or views
 - Run logs of the database system and logs related to cluster management
 - Database system configuration information

Information Collection for One-Click Diagnosis(2)

- Core files generated by database-related processes
- Stack information about database-related processes
- Trace information generated by the database process
- Redo log file XLOG generated by the database
- Planned reproduction information

Slow SQL Diagnosis

- 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. Table-based and function-based APIs help users collect statistics on slow SQL indicators and connect to third-party platforms.
- Slow SQL diagnosis provides information necessary for diagnosing slow SQL statements, helping developers backtrack SQL statements whose execution time exceeded the configured threshold, and diagnose SQL performance bottlenecks.
- 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.

Slow SQL Diagnosis - Feature Enhancement

- 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 dbe_perf.get_global_full_sql_by_timestamp(start_timestamp, end_timestamp);
```

Example:

```
postgres=# 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	coordinator1
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
...	
query_plan	Coordinator Name: coordinator1 Function Scan on pg_show_all_settings a (cost=0.00..12.50 rows=5 width=64) Filter: (name = '***'::text)
...	

Session Performance Diagnosis

- Session diagnosis means diagnosing the 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, session snapshots are used to sample statistical indicators of active sessions. The statistics reflect the basic information, status, and resources of active sessions, including client information, execution start and stop times, SQL text, waiting events, and current database objects. This information can help users diagnose which sessions are consuming 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 can be divided into levels:
 - The first level is real-time information stored in the memory. This includes active session information from the latest several minutes. This is the most precise information available.
 - The second level is the persistent historical information stored to the disks. It includes historical active session data for a long period of time and is sampled from the memory data. This level is suitable for long-run statistics and analysis.

Session Diagnosis Benefits

You can learn:

- what recent events have been consuming the most resources in user sessions.
- which SQL statements have wasting resources waiting events, and for which events.
- which sessions have been wasting resources waiting for events, and for which events.
- information about the users occupying the most resources.
- which sessions have been blocked waiting for other sessions.

Common Statements for Session Performance Diagnosis (1)

- Check the blocking relationships between sessions.

```
select sessionid, block_sessionid from pg_thread_wait_status;
```

- Sample data about blocked sessions.

```
select sessionid, block_sessionid from DBE_PERF.local_active_session;
```

- Display the final blocked session.

```
select sessionid, block_sessionid, final_block_sessionid from DBE_PERF.local_active_session;
```

- Check which wait event consumed 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;
```

Common Statements for Session Performance Diagnosis (2)

- Check which events consumed 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) t
ORDER BY SUM(t.count) OVER (PARTITION BY t.sessionid,
start_time)DESC, t.event;
```

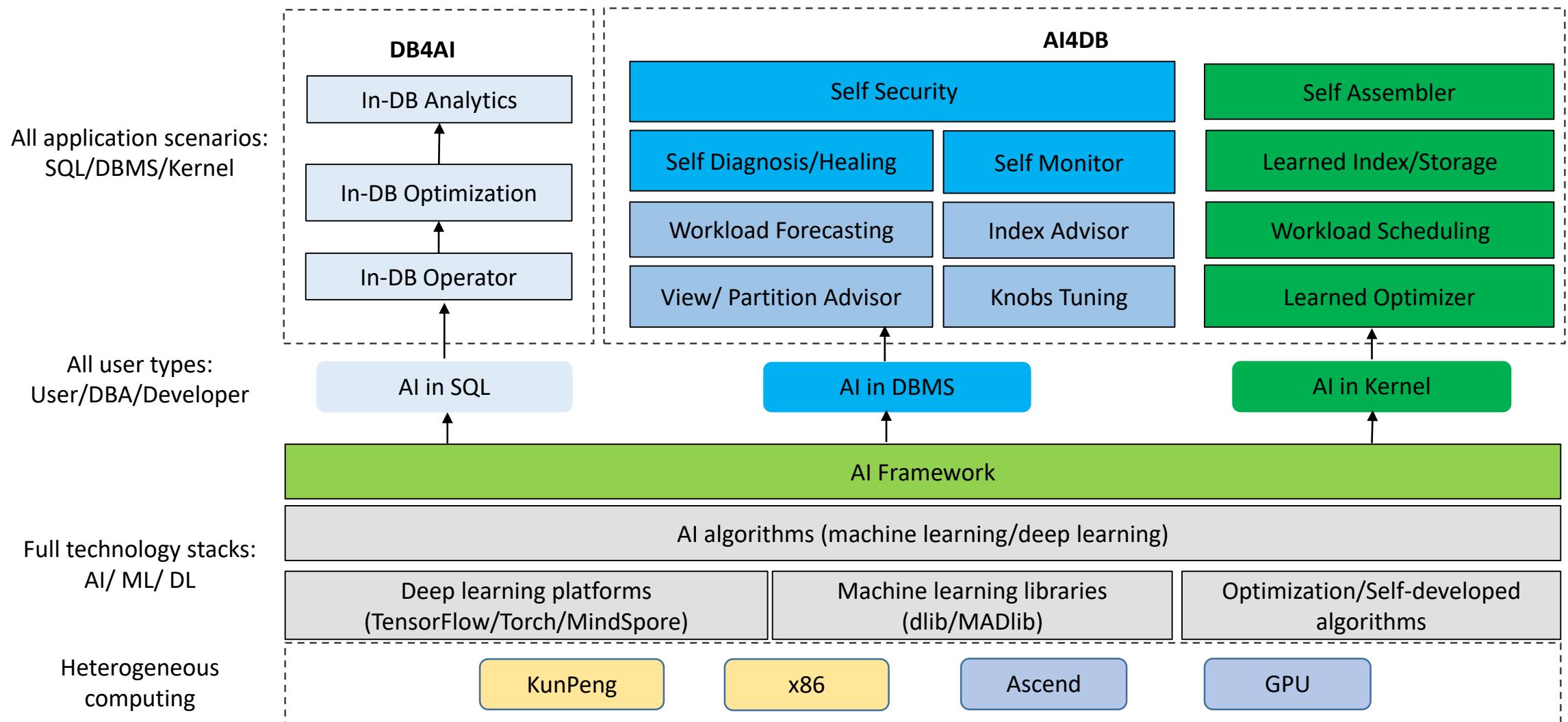
- Check which events that consumed the most resources in the last five minutes.

```
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;
```

Contents

1. System Architectures
 2. Deployment Solutions
 3. Typical Networking
- 4. Key Features**
- High Performance
 - High Availability (HA)
 - High Security
 - Easy Maintenance
- AI Capabilities

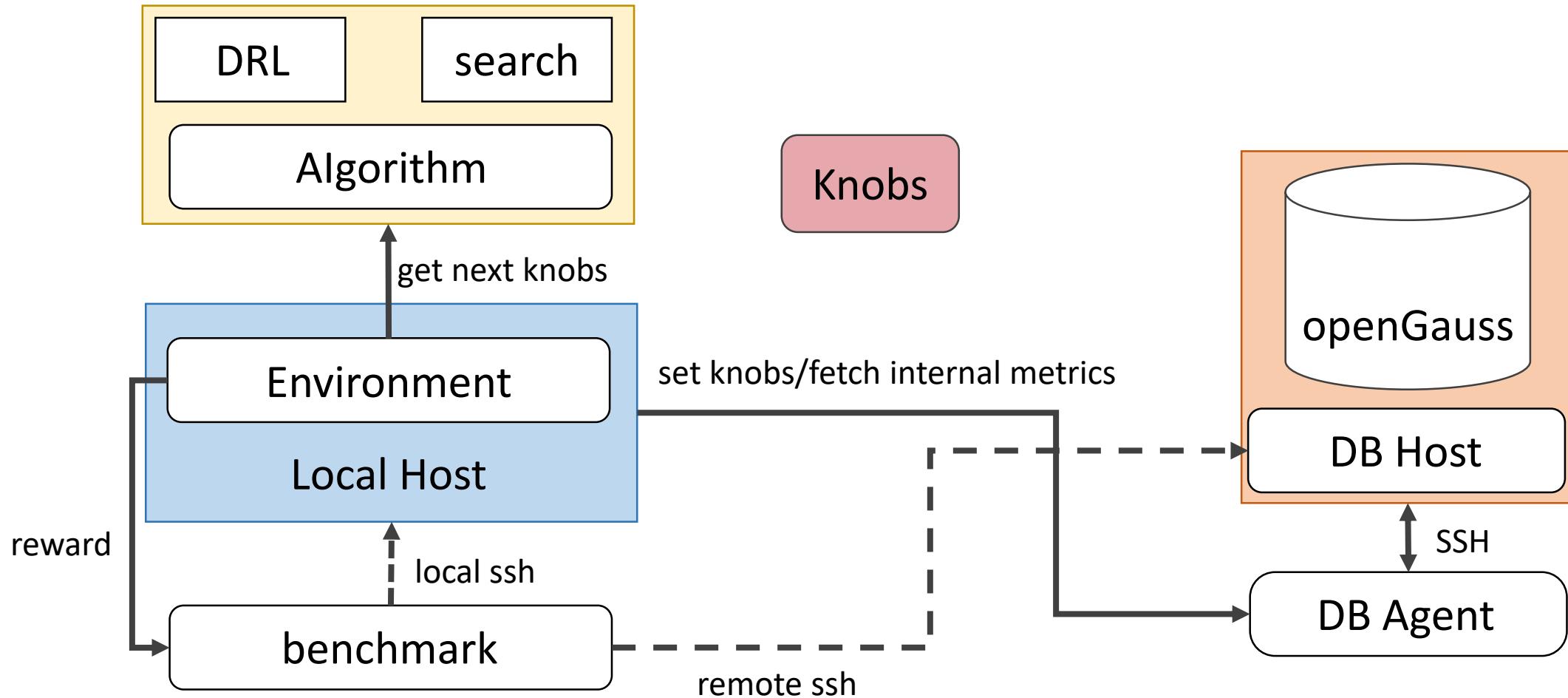
openGauss AI Overview



X-Tuner Overview

- X-Tuner is a parameter tuning tool integrated into databases. It uses AI technologies such as deep reinforcement learning and global search algorithms to obtain the best possible database parameter settings without manual intervention.
- The tuning program is a tool independent from the database kernel. This tool is not necessarily deployed with the database environment. It can be deployed independently and run without the database installation environment. The usernames and passwords for the database and instances are required to control the database benchmark performance testing. Before starting X-tuner, ensure that the interaction in the test environment is normal, the benchmark test script can be run properly, and the database can be connected properly.
- This tool can quickly provide the parameter adjustment configuration of the current load in any scenario, reducing DBA workloads, improving O&M, and meeting customer expectations.

X-Tuner Structure



X-Tuner Modes

- X-Tuner has three modes it can run in:
 - **recommend:** X-tuner logs in to the database using account credentials you specify, obtains information about running workloads, and generates parameter recommendations based on the collected information. It identifies inappropriate parameter settings and potential risks in the current database, and then reports behavior and characteristics of current workloads, along with recommended parameter settings. **In this mode, the database does not need to be restarted. In other modes, the database may need to be repeatedly restarted.**
 - **train:** Parameters are modified based on benchmark information provided by users. A reinforcement learning model is trained through repeated iteration. Later, whenever users run X-tuner in **tune** mode, the model gets optimized.
 - **tune:** Database parameters are tuned based on an optimization algorithm. 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 be run first to generate a model to be tuned. However, the global search algorithm does not need to be trained in advance. It can be directly used for search and optimization.

Installing and Running X-Tuner (1)

- There are two ways to run X-Tuner. One is to run the X-Tuner directly from the source code. The other is to install X-Tuner on the system using Python setup tools, and then run the **gs_xtuner** command to call the X-Tuner.
- Method 1: Run the source code directly.
 - Switch to the **xtuner** source code directory. For the openGauss community code, the path is **openGauss-server/src/gausskernel/dbmind/tools/xtuner**. For an installed database system, the source code path is **\$GAUSSHOME/bin/dbmind/xtuner**.
 - You can view the **requirements.txt** file in the current directory. Use pip to install the dependencies from **requirements.txt**.
 - After the installation is complete, add the environmental variable PYTHONPATH, and then run **main.py**. For example, to obtain
`pip install -r requirements.txt`

```
cd tuner # Switch to the directory where the main.py entry file is located.  
export PYTHONPATH='.' # Add the upper-level directory to the path for searching for packages.  
python main.py --help # Obtain help information. The methods of using other functions are similar.
```

Installing and Running X-Tuner (2)

- Method 2: Install X-Tuner.
 - You can use the **setup.py** file to install X-Tuner and then use **gs_xtuner** to run the program. You need to switch to the **xtuner** root directory: **openGauss-server/src/gausskernel/dbmind/tools/xtuner**. For an installed database system, the location is **\$GAUSSHOME/bin/dbmind/xtuner**.
 - Install Python setup tools:
 - **python setup.py install**
After the installation is completed, the **gs_xtuner** command can be called from anywhere.
 - For example, to obtain the help information, run the following command:

```
gs_xtuner --help
```

SQLdiag Overview

- SQLdiag is an SQL statement diagnostic tool that predicts how long SQL statements will take to execute. It relies on statement logic similarity and historical execution records. There is no need to obtaining the SQL statement execution plan using a template.
- Existing prediction technologies are mainly based on evaluations of the execution plans. This method can only be used in OLAP scenarios where the execution plans can be obtained. They are not very useful for quick simple OLTP or HTAP queries. SQLdiag, in contrast, examines the database SQL statement history and analyzes the performance of these historical statements to predict the performance of new, unknown services. The execution speed of SQL statements database does not vary greatly over a short period of time. That means that as long as there is a similar set of SQL statements in the database history, SQLdiag can predict the speed of the new set based on the historical data. It can predict the execution speed of new SQL statements based on certain templates and on SQL vectorization technology.

SQLdiag Advantages and Prerequisites

- The SQLdiag tool has the following advantages:
 - Execution plans do not require SQL statements. This has no impact on database performance.
 - SQLdiag is a popular tool, unlike industry tools, for example, they are applicable only to OLTP or OLAP.
 - SQLdiag is robust and easy to use. You can train your own prediction model using simple operations.
- The prerequisites for using the SQLdiag tool are as follows:
 - The historical logs and the format of the workload to be predicted need to meet certain 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.

SQLdiag Usage

- The SQLdiag file structure is as follows:

```
Sqldiag
└── data --Test dataset
└── src --Source code file
└── README.md --Description document
└── main.py --Program entry
```

- To collect SQL statements, use the GUC parameters **log_statement** and **log_statement_stats** to enable log collection.
The parameters are as follows:
 - log_statement = all
 - log_statement_stats=on
- Procedure
 - Provide historical logs for model training.
 - Perform training and prediction.

SQLdiag Usage Example

- Use the provided training data for training.

```
python main.py -m train -f data/train.csv
```

- Use the provided test data for prediction.

```
python main.py -m predict -f data/predict.csv
```

- Analyze the results.

```
status: prediction status
data:
    time: SQL statement execution time
    point: spatial point coordinate of an SQL statement
    cluster: SQL type ID
    background: template-based model summary
    stmts: SQL statement sample of the corresponding type
    center: central point coordinate of the SQL statement of the corresponding type
    points: spatial point coordinate of the sample SQL statement
    avg_time: average execution time of the corresponding type
```

Index-advisor Overview

- It provides a pointer to a data value stored in a specified column of a table, similar to a catalog of books, which speeds up the table's query.
- The index recommendation functions include single-query index, virtual index, and workload-level index recommendations.

Single-query Index Recommendation

- Single-query index recommendations allow users to directly perform operations in the database. This function 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.
- This function involves the following API:

Function	Parameter	Description
gs_index_advise	SQL statement string	Generates a recommendation index for a single query statement.

Single-query Index Recommendation Example (1)

- Use **gs_index_advise** to obtain the recommendation index generated for the query. The recommendation results consist of the table name and column name of the index.
- Example 1:

```
postgres=> select * from gs_index_advise('SELECT c_discount from bmsql_customer where c_w_id = 10');
table | column
-----+-----
bmsql_customer | (c_w_id)
(1 row)
```

bmsql_customer table. You can run the following SQL statement to create an index:

```
CREATE INDEX idx on bmsql_customer(c_w_id);
```

Single-query Index Recommendation Example (2)

- Example 2:

```
postgres=# select * from gs_index_advise('select name, age, sex from t1 where age >= 18 and age < 35 and sex = "f";') table | column
-----+-----
t1 | (age, sex)
(1 row)
```

- This statement indicates that a join index (**age, sex**) needs to be created in the **t1** table. You can run the following SQL statement to create an index:

```
CREATE INDEX idx1 on t1(age, sex);
```

Virtual Index

- The virtual index function allows users to directly perform operations in the database. This function 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 GUC parameter **enable_hypo_index** determines whether to enable the virtual index function. The default value is **off**.
- This function involves the following APIs:

Function	Parameter	Description
hypopg_create_index	Character string of the statement for creating an index	Creates a virtual index.
hypopg_display_index	None	Displays information about all created virtual indexes.
hypopg_drop_index	OID of the index	Deletes a specified virtual index.
hypopg_reset_index	None	Clears all virtual indexes.
hypopg_estimate_size	OID of the index	Estimates the space required for creating a specified index.

Virtual Index Example (1)

- Use the **hypopg_create_index** function to create a virtual index, for example:

```
postgres=> select * from hypopg_create_index('create index on bmsql_customer(c_w_id)');
indexrelid | indexname
-----+-----
329726 | <329726>btree_bmsql_customer_c_w_id
(1 row)
```

- Enable the GUC parameter **enable_hypo_index**. This parameter controls whether the database optimizer considers the created virtual index when executing the EXPLAIN statement. By executing **EXPLAIN** on a specific query statement, you can evaluate whether the index can improve the execution efficiency of the query statement based on the execution plan provided by the optimizer, for example:

```
postgres=> set enable_hypo_index = on;
SET
```

- Before enabling the GUC parameter, run **EXPLAIN** and the query statement:

```
postgres=> explain SELECT c_discount from bmsql_customer where c_w_id = 10;
          QUERY PLAN
-----
Seq Scan on bmsql_customer (cost=0.00..52963.06 rows=31224 width=4)
  Filter: (c_w_id = 10)
(2 rows)
```

Virtual Index Example (2)

- After enabling the GUC parameter, run **EXPLAIN** and the query statement.

```
postgres=> explain SELECT c_discount from bmsql_customer where c_w_id = 10;
                                         QUERY PLAN
-----
[Bypass]
Index Scan using <329726>btree_bmsql_customer_c_w_id on bmsql_customer (cost=0.00..39678.69 rows=31224
width=4)
Index Cond: (c_w_id = 10)
(3 rows)
```

- If you compare the two execution plans, you may find that the index may reduce the execution cost of the specified query statement. Then, you can consider creating a real index.

Workload-level Index Recommendations

- For workload-level indexes, you can run scripts outside the database to use this function. This function uses the workload of multiple DML statements as the input to generate a batch of indexes that can improve the overall workload execution performance.
- Prerequisites:
 - The database is normal, and the client can be connected properly.
 - **gsql** has been installed by the current user, and the tool path has been added to the *PATH* environmental variable.
 - The Python 3.6+ environment is available.
- Procedure:
 - Prepare a file that contains multiple DML statements as the input workload. Each statement in the file occupies a line. You can obtain historical service statements from the offline logs of the database.
 - Run the Python script **index_advisor_workload.py**:

```
python index_advisor_workload.py [p PORT] [d DATABASE] [f FILE] [--h HOST] [-U USERNAME] [-W PASSWORD] [--max_index_num  
MAX_INDEX_NUM] [--multi_iter_mode]
```

Workload-level Index Recommendations Example

- Run the **index_advisor_workload.py** script to save multiple DML statements to **tpcc_log.txt**.

```
python index_advisor_workload.py 6001 postgres tpcc_log.txt --max_index_num 10 --multi_iter_mode
```

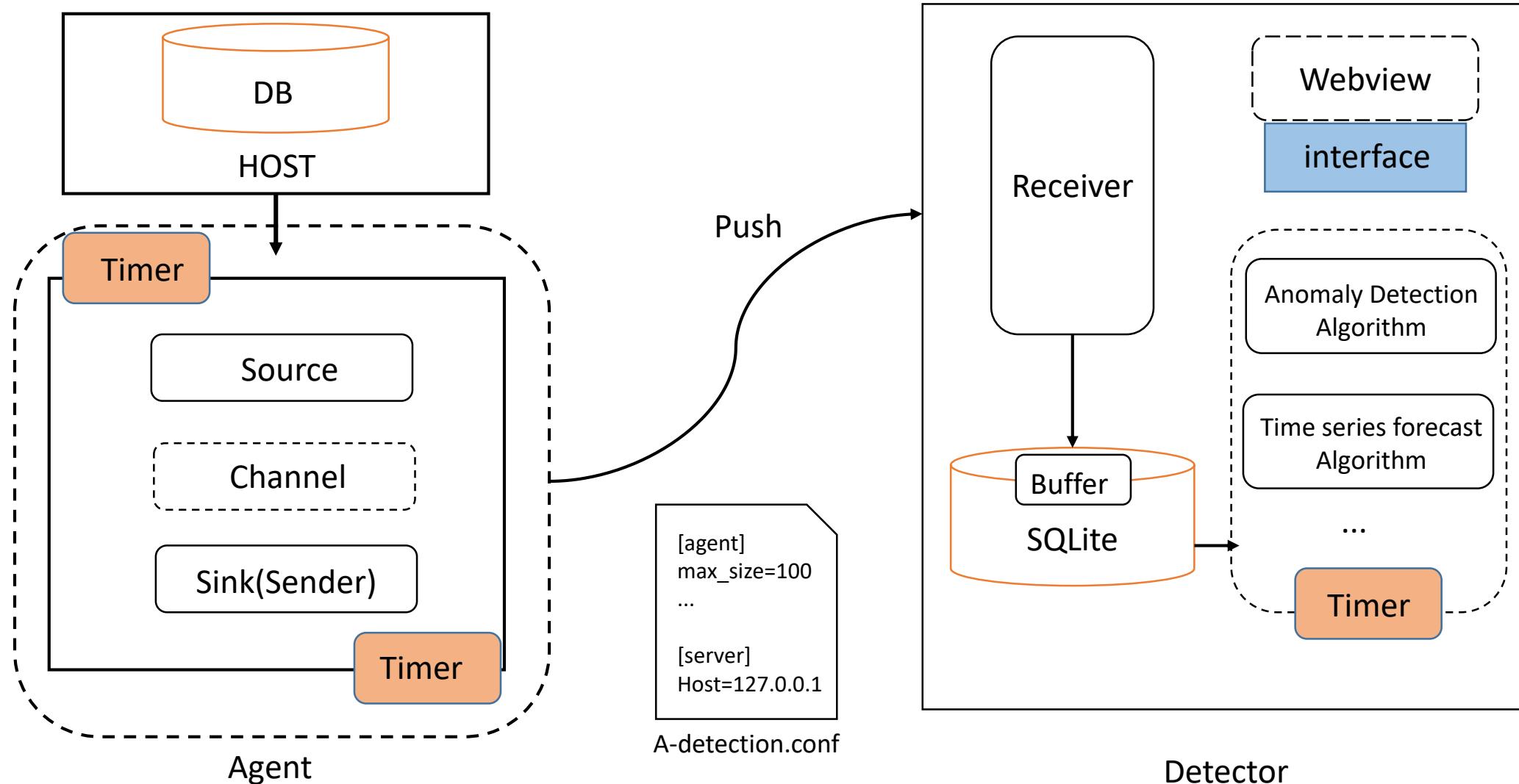
- The recommendation result is a batch of indexes, which are displayed on the screen as multiple create statements:

```
create index ind0 on bmsql_stock(s_i_id,s_w_id);
create index ind1 on bmsql_customer(c_w_id,c_id,c_d_id);
create index ind2 on bmsql_order_line.ol_w_id,ol_o_id,ol_d_id);
create index ind3 on bmsql_item(i_id);
create index ind4 on bmsql_oorder(o_w_id,o_id,o_d_id);
create index ind5 on bmsql_new_order(no_w_id,no_d_id,no_o_id);
create index ind6 on bmsql_customer(c_w_id,c_d_id,c_last,c_first);
create index ind7 on bmsql_new_order(no_w_id);
create index ind8 on bmsql_oorder(o_w_id,o_c_id,o_d_id);
create index ind9 on bmsql_district(d_w_id);
```

Anomaly_detection Overview

- Anomaly_detection is an AI tool integrated into openGauss and can be used to collect and predict database indicators, as well as monitor and diagnose exceptions. It is a component in the dbmind suite. It can collect indicators for IO_Read, IO_Write, CPU_Usage, Memory_Usage, and disk space occupied by the database. Anomaly_detection can monitor multiple indicators at the same time and predict the trends for each indicator. If it detects an indicator has exceeded the configured threshold in a certain period or predicts it will at some point in the future, the tool generates an alarm through logs.
- Anomaly_detection consists of an agent and a detector.
 - The agent and the openGauss database are deployed on the same server.
 - The detector module communicates with the agent module over HTTP or HTTPS. The detector module can be deployed on any server that can communicate with the agent module.

Anomaly_detection Structure



Installation and Running of anomaly_detection

- Switch to the **anomaly_detection** directory. For the openGauss community code, it is **openGauss-server/src/gausskernel/dbmind/tools/anomaly_detection**. For the installed database system, the source code path is **\$GAUSSHOME/bin/dbmind/anomaly_detection**.
- You can view the **requirements.txt** file in the current directory. Use pip to install the dependencies from **requirements.txt**.

```
pip install -r requirements.txt
```

- After the installation is complete, run **main.py**. For example, to access the help file:

```
cd anomaly_detection # Switch to the directory where the main.py entry file is located.  
python main.py --help # Obtain help information. The methods of using other functions are similar.
```

Adding Monitoring Parameters

- Write an API for obtaining metric data. Take io_read as an example. Write the following io_read metrics obtaining a function in **metric_task.py** under the **task** directory:

```
def io_read():
    child1 = subprocess.Popen(['pidstat', '-d'], stdout=subprocess.PIPE, shell=False)
    child2 = subprocess.Popen(['grep', 'gaussd[b]'], stdin=child1.stdout, stdout=subprocess.PIPE, shell=False)
    result = child2.communicate()
    if not result[0]:
        return 0.0
    else:
        return result[0].split()[3].decode('utf-8')
```

- Add the io_read section to **metric_task.conf** under the **task** directory.

```
[io_read]
minimum = 30
maximum = 100
data_period = 1H
forecast_interval = 2H
forecast_period = 30M
```

Restart the service.

- Restart the agent.

```
# Local server:  
  python main.py stop --role agent  
  python main.py stop --role agent  
# Remote server:  
  python main.py start --user USER --host HOST --project-path PROJECT_PATH --role agent  
  python main.py stop --user USER --host HOST --project-path PROJECT_PATH --role agent
```

- Restart the server.

```
# Local server:  
  python main.py stop --role server  
  python main.py stop --role server  
# Remote server:  
  python main.py start --user USER --host HOST --project-path PROJECT_PATH --role server  
  python main.py stop --user USER --host HOST --project-path PROJECT_PATH --role server
```

Predictor Overview

- Predictor is a query time prediction tool that leverages machine learning and online learning capabilities. By continuously analyzing historical execution information collected in the database, Predictor can predict how long a plan will take to execute.
- To use this tool, you must start the Python process AI Engine for model training and inference.
- Prerequisites:
 - openGauss is normal and users can log in to openGauss.
 - The SQL syntax is correct, no errors are reported, and there are not database exceptions.
 - In the historical performance data window, the openGauss concurrency is stable, the table structure and table quantity remain unchanged, the data volume does not change abruptly, and the GUC parameters related to query performance remain unchanged.
 - During prediction, the model needs to have been trained and converged.
 - The operating environment of the AiEngine is stable.

Data Collection (1)

- Enable data collection.

- Set parameters related to the ActiveSQL operator.

```
enable_resource_track=on  
resource_track_level=operator  
enable_resource_record=on  
resource_track_cost=10 # The default value is 100000.
```

- Collect information and run the service query statement to view the real-time collection data. All the jobs that meet the requirements of **resource_track_duration** and **resource_track_cost** should be collected.

```
select * from gs_wlm_plan_operator_history;
```

- Disable data collection.

- Set parameters related to the ActiveSQL operator.

```
enable_resource_track=off, resource_track_level=none, or resource_track_level=query
```

- Execute the service query statement. Wait for 3 minutes and check the data on the current node. No new data should be added to the table or view.

```
select * from gs_wlm_plan_operator_info;
```

Data Collection (2)

- Store data persistently.
 - Set parameters related to the ActiveSQL operator.

```
enable_resource_track=on
resource_track_level=operator
enable_resource_record=on
resource_track_duration=0 # The default value is 60s.
resource_track_cost=10 # The default value is 100000.
```

- Execute the service query statement. Wait for 3 minutes and check the data on the current node.
(Expected result: All jobs that meet the requirements of **resource_track_duration** and
resource_track_cost are collected.)

```
select * from gs_wlm_plan_operator_info;
```

Model Management

- Add a model. `INSERT INTO gs_opt_model values('.....');`

- Example:

```
INSERT INTO gs_opt_model values('rlstm', 'model_name', 'datname', '127.0.0.1', 5000, 2000, 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

- Modify model parameters.

```
UPDATE gs_opt_model SET <attribute> = <value> WHERE model_name = <target_model_name>;
```

- Delete a model.

```
DELETE FROM gs_opt_model WHERE model_name = <target_model_name>;
```

- Query the existing model and its status.

```
SELECT * FROM gs_opt_model;
```

Model Training

- Configure or add model training parameters.

```
INSERT INTO gs_opt_model values('rlstm', 'default', 'postgres', '127.0.0.1', 5000, 2000, 1, -1, 64, 512, 0 , false, false, '{S, T}', '{0,0}', '{0,0}', 'Text');
```

- Delete the original encoding data on the premise that the database status is normal and historical data is collected properly.

```
DELETE FROM gs_wlm_plan_encoding_table;
```

- To encode data, specify the database name.

```
SELECT gather_encoding_info('postgres');
```

- Start training.

```
SELECT model_train_opt('rlstm', 'default');
```

- View the model training status.

```
SELECT * FROM track_model_train_opt('rlstm', 'default');
```

DeepSQL Overview (1)

- The database DeepSQL feature uses DB4AI, and AI algorithm built in to the database for faster analysis and processing of big data. openGauss provides a complete set of SQL-based machine learning, data mining, and statistics algorithms. Users can use SQL statements for machine learning.
- DeepSQL can abstract end-to-end R&D processes from data to models. With a bottom-layer engine and automatic optimization, technical personnel with basic SQL knowledge can complete most machine learning model training and prediction tasks. The entire analysis and processing run on the database engine. Users can directly analyze and process data in the database. There is no need to transfer data between the database and other platforms. The need to move data between multiple environments is eliminated.

DeepSQL Overview (2)

- DeepSQL is an enhanced version of openGauss DB4AI. DeepSQL encapsulates common machine learning algorithms into SQL statements and supports more than 60 general algorithms, including regression algorithms (such as linear regression, logistic regression, and random forest), classification algorithms (such as KNN), and clustering algorithms (such as K-means). In addition to basic machine learning algorithms, graph-related algorithms are also included, such as algorithms about the shortest path and graph diameter. Also, it supports data processing (such as PCA), sparse vectors, common statistical algorithms (such as covariance and Pearson coefficient calculation), training set and test set segmentation, and cross validation.
- The DeepSQL environment consists of two parts: compiling the database and installing the algorithm library.
- Prerequisites:
 - Python 2.7.12 or later has been installed.
 - The database supports the PL/Python stored procedure.
 - You have the administrator permission to install the algorithm library.

Quiz

1. (Multiple-answer question) Which of the following are the openGauss deployment modes?
 - A. Standalone deployment
 - B. Primary/Standby
 - C. One primary and multiple standbys deployment
 - D. Primary-primary deployment
2. (Short answer question) What are the two components of the openGauss AI feature? What are the features of each component?

Summary

- This chapter describes the physical architecture and logical architecture of the openGauss database, and describes the typical networking and deployment scheme of the openGauss database. In addition, this chapter describes the key features of openGauss database in terms of high performance, high availability, high security, easy maintenance, and AI capabilities.

Recommendations

- HUAWEI CLOUD:
 - <https://www.huaweicloud.com/intl/en-us/>
- openGauss help document:
 - <https://opengauss.org/en/docs/2.0.0/docs/installation/installation.html>
- openGauss code repository:
 - <https://gitee.com/opengauss>
- GaussDB(for openGauss) help document:
 - https://support.huaweicloud.com/intl/en-us/productdesc-opengauss/opengauss_01_0002.html

Acronyms and Abbreviations

- MOT: memory-optimized table
- WAL: write-ahead log
- OM: Operation Manager
- RBO: rule-based optimization
- CBO: cost-based optimization
- MCV: most_common_vals
- NUMA: non-uniform memory access
- SMP: symmetric multi-processing
- MPP: massively parallel processing
- WDR: workload diagnosis report
- GE: Gigabit Ethernet
- DR: disaster recovery

04 Database and Object Management



Foreword

- openGauss is an open-source relational database management system released under the Mulan Permissive Software License v2. The openGauss kernel is derived from PostgreSQL. It deeply integrates Huawei's years of experience in the database field and continuously builds competitive features based on enterprise-level scenario requirements.
- This chapter describes the openGauss logical structure, storage engine selection, tablespace creation and management, user and role management, system catalogs and views, data import and export, and high-risk operations.

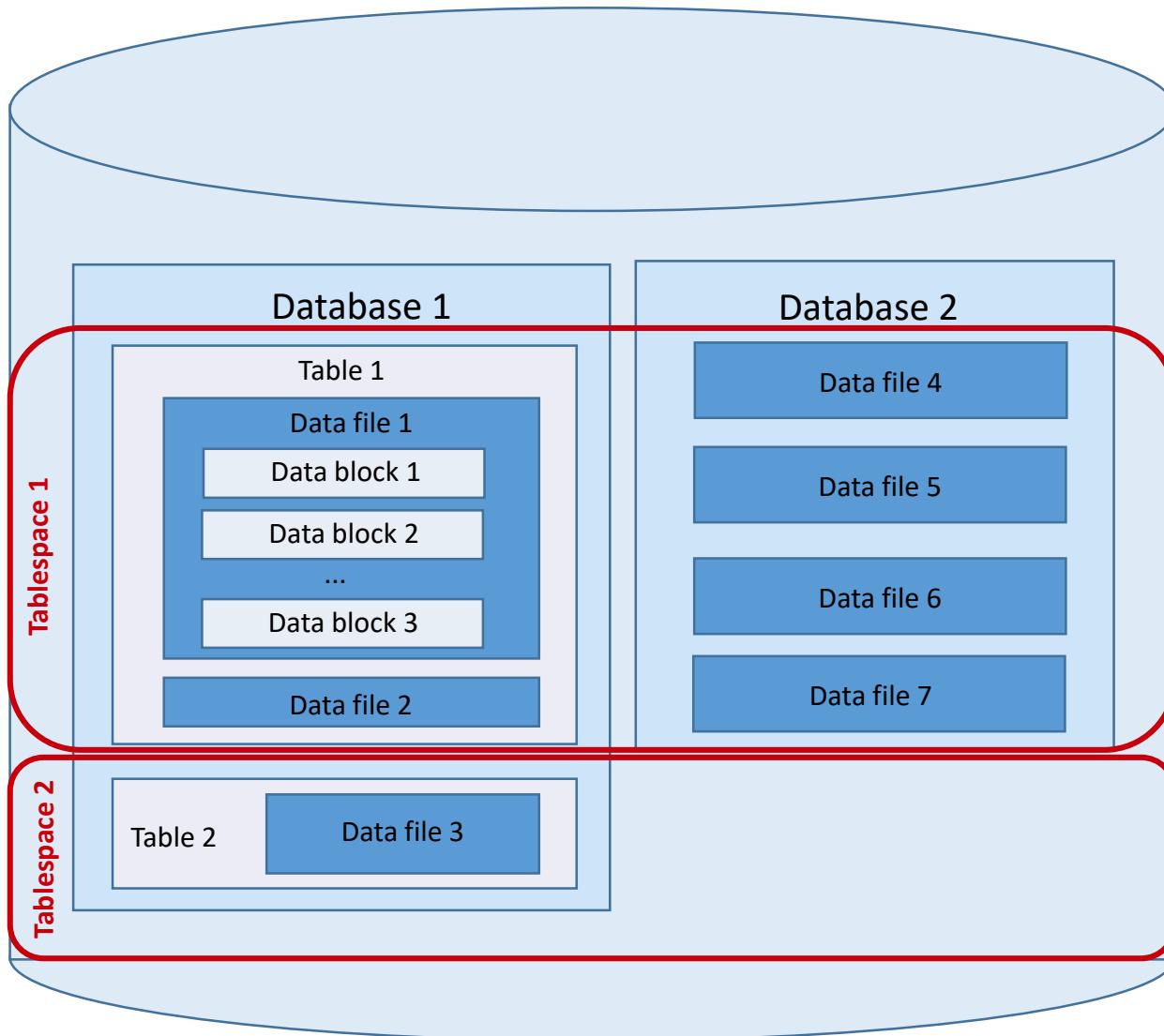
Objectives

- On completion of this course, you will be able to:
 - Understand the openGauss logical structure.
 - Master how to select different storage engines.
 - Master how to create and manage tablespaces.
 - Master how to manage users and roles.
 - Understand common system catalogs and views.
 - Master how to import and export data.
 - Understand high-risk operations of openGauss.

Contents

- 1. Logical Structure**
2. Storage Engine
3. Tablespace
4. User Roles
5. System Catalogs and System Views
6. Data Import and Export
7. High-Risk Operations

openGauss Logical Architecture



Concepts

- Tablespace: Directory storing physical files of its databases. Multiple tablespaces can coexist, and each of them can contain files belonging to different databases.
- Database: Databases manage various data objects and are isolated from each other. Objects managed by a database can be distributed to multiple tablespaces.
- Datafile Segment: Each datafile stores the data of only one table. A table containing more than 1 GB of data is stored in multiple datafile segments.
- Table: Each table belongs to only one database and one tablespace. Datafile segments that store the data of the same table must be in the same tablespace.
- Block: Basic unit of database management. Its default size is 8 KB.

Contents

1. Logical Structure
- 2. Storage Engine**
3. Tablespace
4. User Roles
5. System Catalogs and System Views
6. Data Import and Export
7. High-Risk Operations

Storage Models

- openGauss supports the hybrid row-column store. Each storage mode applies to specific scenarios. Select an appropriate mode when creating a table. Generally, openGauss is used for OLTP databases. By default, the row store is used. The column store is used only in OLAP scenarios where complex queries in large data volume are performed.
- openGauss uses a Memory-Optimized Table (MOT) to provide a higher-performance load for transactional work. MOT has significant advantages in terms of high performance (query and transaction latency), high scalability (throughput and concurrency), and even cost (high resource utilization) in some cases.

Row-Store Table

- Tables are stored to disk partitions by row and are called row-store tables. The following shows a row-store table on a hard disk, with content being stored by row.

Row ID	Time/D ate	Material	Customer Name	Quantity
1	845	2	3	1
2	851	5	2	2
3	872	4	4	1
4	878	1	5	2
5	888	2	3	3
6	895	3	4	1
7	901	4	1	1

1	845	2	3	1	2	851	5	2	2	3	872	4	4	1	4	878	1	5	2
---	-----	---	---	---	---	-----	---	---	---	---	-----	---	---	---	---	-----	---	---	---

Row-Store Table Example

- Row-store tables are created by default. Data is continuously stored by row. Therefore, this storage model applies to scenarios where data needs to be updated frequently.
- Example:

```
postgres=# CREATE TABLE customer_t1
(
    state_ID  CHAR(2),
    state_NAME VARCHAR2(40),
    area_ID   NUMBER
);

-- Delete the table.
postgres=# DROP TABLE customer_t1;
```

Column-Store Table

- Tables are stored to disk partitions by column. These tables are column-store tables. The following shows a column-store table on a hard disk, with the content being stored by column.

Row ID	Time/Date	Material	Customer Name	Quantity
1	845	2	3	1
2	851	5	2	2
3	872	4	4	1
4	878	1	5	2
5	888	2	3	3
6	895	3	4	1
7	901	4	1	1

Diagram illustrating the storage of a column-store table on disk. Red vertical arrows point from each column header to its corresponding column of data values at the bottom. The columns are: Row ID, Time/Date, Material, Customer Name, and Quantity. The data values are: Row ID (1 to 7), Time/Date (845 to 901), Material (2, 5, 4, 1, 2, 3, 4), Customer Name (3, 2, 4, 5, 3, 4, 1), and Quantity (1, 2, 1, 2, 3, 1, 1).

1 2 3 4 ... 845 851 872 878 ... 2 5 4 1 ... 3 2 4 5 ...

Column-Store Table Example

- Column-store tables need to be declared upon creation. In a column-store table, data is stored by column, with data in each column being stored continuously. The I/O of a single-column query is small and occupies less storage space than that of a row-store table. This storage model applies to scenarios where data is inserted in batches, is less frequently updated, and is queried for statistical analysis. However, column-store tables are not suitable for point queries.
- Example:

```
postgres=# CREATE TABLE customer_t2
(
    state_ID  CHAR(2),
    state_NAME VARCHAR2(40),
    area_ID   NUMBER
)
WITH (ORIENTATION = COLUMN);

-- Delete the table.
postgres=# DROP TABLE customer_t2;
```

Comparison Between Row-Store and Column-Store Tables

Storage Model	Advantage	Disadvantage	Scenario
Row store	Data is stored together. INSERT and UPDATE operations are efficient.	All the columns of a record are read after the SELECT statement is executed even if only certain columns are required.	Point queries (simple index-based queries that only return a few records). Scenarios requiring frequent addition, deletion, and modification.
Column store	Only the columns involved in a query are read. Projections are efficient. Any column can serve as an index.	The selected columns need to be reconstructed after the SELECT statement is executed. INSERT and UPDATE are complex.	Statistical analysis queries (requiring a large number of association and grouping operations). Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables).

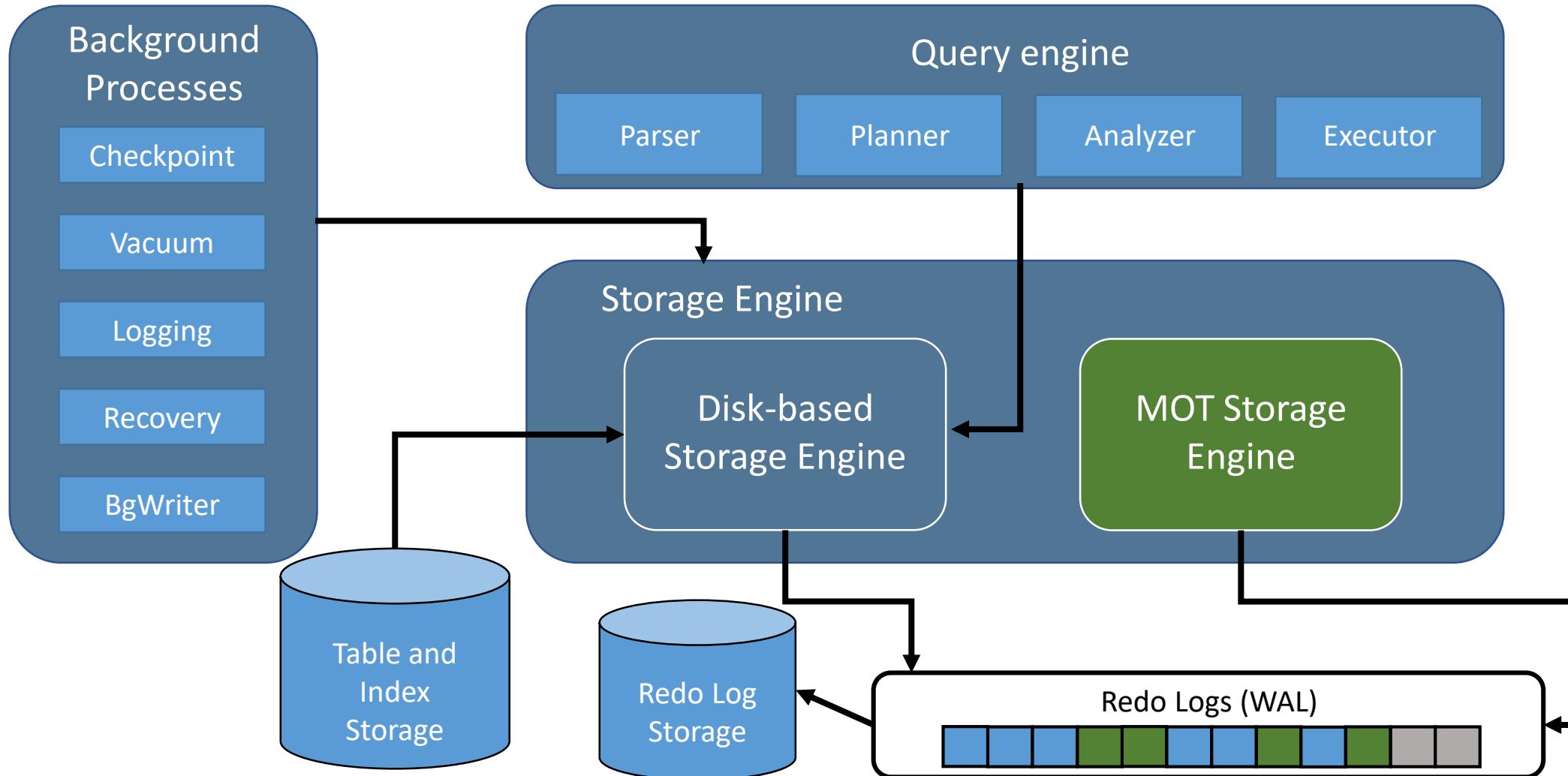
Storage Model Selection

- Update frequency
 - If data is frequently updated, use a row-store table.
- Data insert frequency
 - If data is frequently inserted and the data volume is small 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 many columns are required, use a column-store table.
- Number of columns to be queried
 - If few columns in a table are required (less than 50% of the total number of columns) in each query, use a column-store table.
- Compression ratio
 - A column-store table has a higher compression rate than a row-store table, but a high compression rate consumes more CPU resources.

MOT

- openGauss uses the MOT storage engine, which is a transactional row storage and is optimized for multi-core and large-memory servers. MOT is the most advanced production-level feature of openGauss databases. It provides higher performance for transactional workloads. Enterprises can use MOT in mission-critical and performance-sensitive online transaction processing (OLTP) to achieve high performance, high throughput, predictable low latency, and high utilization of multi-core servers. MOT and disk-based common tables are created side by side. The effective design of MOT implements almost complete SQL coverage and supports a complete set of database functions, such as stored procedures and user-defined functions.
- MOT fully supports ACID features and includes strict persistence and high availability support.
- MOT provides faster data access and more efficient transaction execution through data and indexes completely stored in memory, non-uniform memory access-aware (NUMA-aware) design, algorithms that eliminate locks and lock contention, and query native compilation.
- The lock-free design and highly optimized implementation of MOT enables it to achieve excellent near-linear throughput expansion on multi-core servers.

openGauss Structure



MOT Features and Benefits

- MOT has significant advantages in terms of high performance (query and transaction latency), high scalability (throughput and concurrency), and even cost (high resource utilization) in some cases.
 - Low latency: provides fast query and transaction response time.
 - High throughput: supports peak and continuous high user concurrency.
 - High resource utilization: fully utilizes hardware.
- Applications that use MOT can reach 2.5 to 4 times the throughput compared to applications that do not use MOT. For example, after the TPC-C benchmark test is performed on the Huawei TaiShan servers based on ARM/Kunpeng processors, MOT provides 2.5x throughput gains on 2-socket servers and 4.8 million tpmC on 4-socket 256-core Arm servers.
- In the TPC-C benchmark test, you can observe that MOT provides lower latency and reduces the transaction speed by 3 to 5.5 times.
- The high load and high contention situation is a recognized problem for all industry-leading databases, and MOT can make extremely high use of server resources in this situation. After MOT is used, the resource utilization of 4-socket servers reaches 99%.

MOT Usage

- The memory configuration must meet the following requirements: $(\text{max_mot_global_memory} + \text{max_mot_local_memory}) + 2 \text{ GB} < \text{max_process_memory}$
 - Global memory is long-term memory that is shared by all cores and is used primarily to store all table data and indexes.
 - Local memory is short-term memory that is used primarily by sessions for handling transactions and store data changes in prime to transaction memory until the commit phase.
- To enable a specific user to create and access MOT (DDL, DML, SELECT), execute the following statement only once:

```
GRANT USAGE ON FOREIGN SERVER mot_server TO <user>;
```

- Create an MOT.

```
create FOREIGN table test(x int) [server mot_server];
```
- Delete an MOT.

```
drop FOREIGN table test;
```
- Create an index for an MOT.

```
create index text_index1 on test(x);
```
- Convert a disk table to MOT: Use the gs_dump tool to dump the disk table data to the physical file of the disk, and then use the gs_restore tool to load or restore the data from the disk file to the database table.

Contents

1. Logical Structure
2. Storage Engine
- 3. Tablespace**
4. User Roles
5. System Catalogs and System Views
6. Data Import and Export
7. High-Risk Operations

Tablespace

- The administrator can use tablespaces to control the layout of disks where a database is installed. The following are the advantages:
 - If the disk partition or volume allocated to the initialized database is full and the space cannot be logically increased, you can create and use tablespaces in other partitions until the space is reconfigured.
 - Tablespaces allow the administrator to distribute data based on the schema of database objects, improving system performance.
 - A frequently used index can be placed in a disk with stable performance and high computing speed.
 - A table that stores archived data and is rarely used or has low performance requirements can be placed in a disk with a slow computing speed.
 - You can use tablespaces to control the disk space occupied by data in a database. If the usage of a disk where a tablespace resides reaches 90%, the database switches to read-only mode. It switches back to read/write mode when the disk usage becomes less than 90%.
 - A tablespace corresponds to a file system directory, and you must have the read and write permissions on an empty directory.

Tablespace Example (1)

- Create a tablespace.

```
postgres=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
```

- **fastspace** indicates the created tablespace.
 - **/tablespace/tablespace_1** is the data directory relative to the database node. It is an empty directory in which you have the read and write permissions.

- A database system administrator can run the following command to grant the permission of accessing the **fastspace** tablespace to user **jack**:

```
postgres=# GRANT CREATE ON TABLESPACE fastspace TO jack;
```

Tablespace Example (2)

- Create an object in a tablespace.
 - If you have the CREATE permission on the tablespace, you can create database objects in the tablespace.
 - Method 1: Create a table in the specified tablespace.

- postgres=# CREATE TABLE foo(i int) TABLESPACE fastspace;

```
postgres=# SET default_tablespace = 'fastspace';
postgres=# CREATE TABLE foo2(i int);
```

- Check the tablespace.
 - Method 1: Check the **pg_tablespace** system catalog.
 - Method 2: Run the gsql meta-command **\db** to check.

Tablespace Example (3)

- Check the tablespace usage.

- Query the current usage of the tablespace.

```
postgres=# SELECT PG_TABLESPACE_SIZE('example');
```

- Calculate the tablespace usage using the following formula: Tablespace usage = PG_TABLESPACE_SIZE/Size of the disk where the tablespace directory is located.

- Modify the tablespace.

- Run the following command to rename tablespace **fastspace** to **fspace**:

```
postgres=# ALTER TABLESPACE fastspace RENAME TO fspace;
```

- Delete the tablespace.

```
postgres=# DROP TABLESPACE fspace;
```

Contents

1. Logical Structure
2. Storage Engine
3. Tablespace
- 4. User Roles**
5. System Catalogs and System Views
6. Data Import and Export
7. High-Risk Operations

User

- User
 - A user is an individual who uses the database service.
 - A user has responsibilities and is often affiliated to an organization or department.
- Database users can:
 - Use a tool to connect to the database.
 - Access database objects.
 - Execute SQL statements.

Role

- Role
 - A role is a set of users. Roles with different permissions are divided based on the responsibilities in the database system. A role is a carrier of permission sets.
 - A role represents the behavioral constraint of a database user or a group of data users.
- openGauss provides an implicitly defined group **PUBLIC** that contains all roles. By default, all new users and roles have the permissions of **PUBLIC**.
- To revoke permissions of **PUBLIC** from a user or role, or re-grant these permissions to them, add the **PUBLIC** keyword in the **REVOKE** or **GRANT** statement.

Users and Roles

- A user is an entity, and a role is a behavior.
- A user is assigned one or more roles.
- A role is a set of permissions. It should not have permission to log in to the database and execute SQL statements.
- User permission management is simplified to role permission management.
- In openGauss, users and roles use the same operation and maintenance methods.

User and Role Operations (1)

- For creation operations, view the **CREATE USER** syntax. (**CREATE ROLE** is the same as **CREATE USER**.)

```
postgres=# \help create user
Command: CREATE USER
Description: define a new database role
Syntax:
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' |
DISABLE };

where option can be:
{SYSADMIN | NOSYSADMIN}
| {AUDITADMIN | NOAUDITADMIN} | {CREATEDB | NOCREATEDB} | {USEFT | NOUSEFT}
| {CREATEROLE | NOCRAEROLE} | {INHERIT | NOINHERIT} | {LOGIN | NOLOGIN}
| {REPLICATION | NOREPLICATION} | {INDEPENDENT | NOINDEPENDENT} | {VCADMIN | NOVCADMIN}
| CONNECTION LIMIT connlimit | VALID BEGIN 'timestamp' | VALID UNTIL 'timestamp'
| RESOURCE POOL 'respool' | USER GROUP 'groupuser' | PERM SPACE 'spacelimit'
| NODE GROUP logic_cluster_name | IN ROLE role_name [, ...] | IN GROUP role_name [, ...]
| ROLE role_name [, ...] | ADMIN role_name [, ...] | USER role_name [, ...]
| SYSID uid | DEFAULT TABLESPACE tablespace_name | PROFILE DEFAULT
| PROFILE profile_name | PGUSER
```

User and Role Operations (2)

- For modification operations, view the **ALTER USER** syntax.

```
postgres=# \help alter user
Command: ALTER USER
Description: change a database role
Syntax:
ALTER USER user_name [ [ WITH ] option [ ... ] ];
ALTER USER user_name
    RENAME TO new_name;
ALTER USER user_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT }|FROM CURRENT};
ALTER USER user_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};

where option can be:
{CREATEDB | NOCREATEDB}
| {CREATEROLE | NOCREATEROLE} | {INHERIT | NOINHERIT} | {AUDITADMIN | NOAUDITADMIN}
| {SYSADMIN | NOSYSADMIN} | {USEFT | NOUSEFT} | {LOGIN | NOLOGIN} | {REPLICATION | NOREPLICATION}
| {INDEPENDENT | NOINDEPENDENT} | {VCADMIN | NOVCADMIN} | CONNECTION LIMIT connlimit
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE }
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] | DISABLE }
| VALID BEGIN 'timestamp' | VALID UNTIL 'timestamp' | RESOURCE POOL 'respool'
| USER GROUP 'groupuser' | PERM SPACE 'spacelimit' | NODE GROUP logic_cluster_name
| ACCOUNT { LOCK | UNLOCK } | PGUSER
```

User and Role Operations (3)

- Query
 - The openGauss user and role attributes are maintained in the **pg_authid** system catalog. You can also query the user and role information in the **pg_user** and **pg_roles** views.
 - Run the **\d** command to view the **pg_authid** details.
 - Run the **select** command to query information in **pg_authid**.

```
postgres=# \d pg_authid
postgres=# select * from pg_authid;
```

User and Role Operations (4)

- Delete
 - View the **DROP USER** syntax.
 - If a user has database objects, **RESTRICT** is used by default and the user cannot be deleted. In this case, the **CASCADE** option is required.

```
postgres=# \help drop user
Command:  DROP USER
Description: remove a database role
Syntax:
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

```
postgres=# \help drop role
Command:  DROP ROLE
Description: remove a database role
Syntax:
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

Set User Account Validity Period

- When creating a user, you need to specify the validity period of the user, including the start time and end time.
- To enable a user not within the validity period to use an account, set a new validity period.
 - Run the following command to create a user and specify the start time and end time of the account's validity period:

```
CREATE USER joe WITH PASSWORD 'Bigdata@123' VALID BEGIN '2015-10-10 08:00:00' VALID UNTIL '2016-10-10 08:00:00';
```

- If the user is not within the specified validity period, run the following command to set the start time and end time of a new validity period:

```
ALTER USER joe WITH VALID BEGIN '2016-11-10 08:00:00' VALID UNTIL '2017-11-10 08:00:00';
```

- Note: If **VALID BEGIN** is not specified in the **CREATE ROLE** or **ALTER ROLE** statement, the start time of the validity period is not limited. If **VALID UNTIL** is not specified, the end time of the validity period is not limited. If both of the parameters are not specified, the user is always valid.

Manually Locking and Unlocking Accounts

- Run the following commands to manually lock and unlock the **user_read** user.
- To manually lock the account:

```
ALTER USER user_read ACCOUNT LOCK;
```

- If the following information is displayed, the account has been successfully locked:

```
ALTER ROLE
```

- To manually unlock the account:

```
ALTER USER user_read ACCOUNT UNLOCK;
```

- If the following information is displayed, the account has been successfully unlocked:

```
ALTER ROLE
```

Private User

- Background
 - If multiple service departments use different database user accounts to perform service operations and a database maintenance department at the same level uses administrator user accounts to perform maintenance operations, service departments require that database administrators, without specific authorization, can manage (**DROP**, **ALTER**, and **TRUNCATE**) but cannot access (**INSERT**, **DELETE**, **UPDATE**, **SELECT**, and **COPY**) the data. That is, the control permissions of database administrators for tables need to be isolated from their access permissions to improve the data security for normal users.
 - In separation-of-duties mode, a database administrator does not have permission to access the tables in schemas of other users. In this case, database administrators have neither management permissions nor access permissions, which does not meet the requirements of the service departments mentioned above. Therefore, openGauss makes use of private users to solve the problem. That is, private users with the **INDEPENDENT** attribute in non-separation-of-duties mode would have to be created.
- Definition:
 - `CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY "1234@abc";` **access (INSERT, DELETE, SELECT, UPDATE, COPY, GRANT, REVOKE, and ALTER OWNER)** the objects without authorization.

Contents

1. Logical Structure
2. Storage Engine
3. Tablespace
4. User Roles
- 5. System Catalogs and System Views**
6. Data Import and Export
7. High-Risk Operations

Overview of System Catalogs and System Views

- System catalogs store the structured metadata of openGauss. They are the source of information used by openGauss to control system running and are a core component of the database system.
- System views provide ways to query the system catalogs and internal database status.
- System catalogs and system views are visible to either system administrators or all users. Some system catalogs and views have marked the need of administrator permissions, so they are accessible only to administrators.
- You can delete and re-create system catalogs, add columns to them, and insert and update values in them, but doing so may make system information inconsistent and cause system faults. Generally, users should avoid modifying system catalogs or system views, or renaming their schemas. They are automatically maintained by the system.

Common System Catalogs (1)

System Catalog Name	Description
GS_WLM_USER_RESOURCE_HISTORY	Stores information about resources used by users and is valid only on CNs. Each record in this system catalog indicates resource usage of a user at a point in time, including the memory, number of CPU cores, storage space, temporary space, operator flushing space, logical I/O traffic, number of logical I/O operations, and logical I/O rate. The memory, CPU, and I/O monitoring items record only resource usage of complex jobs. For I/O monitoring items, this parameter is valid only when enable_logical_io_statistics is set to on . The function of saving user monitoring data is enabled only when enable_user_metric_persistent is set to on .
PG_AM	Stores information about index access methods. There is one row for each index access method supported by the system.
PG_ATTRIBUTE	Stores information about table columns.
PG_AUTHID	Stores information about database authorization identifiers (roles). The concept of users is contained in that of roles. A user is actually a role whose rolcanlogin has been set. Any role, whether its rolcanlogin is set or not, can use other roles as members. For a cluster, only one pg_authid exists which is not available for every database. This system catalog is accessible only to system administrators.
PG_CLASS	Stores information about database objects and their relationships.
PG_CONSTRAINT	Stores check, primary key, unique, and foreign key constraints on tables.
PG_DEPEND	Records dependencies among database objects. This information allows DROP commands to find which other objects must be dropped by DROP CASCADE or prevent dropping in the DROP RESTRICT case.
PG_DATABASE	Stores information about the available databases.

Common System Catalogs (2)

System Catalog Name	Description
PG_FOREIGN_TABLE	Stores auxiliary information about foreign tables.
PG_INDEX	Stores part of the information about indexes. The rest is mostly stored in PG_CLASS .
PG_JOB	Stores detailed information about jobs created by users. Dedicated threads poll the PG_JOB table and trigger jobs based on scheduled job execution time, and update job status in the PG_JOB table. This system catalog belongs to the Shared Relation category. All job records are visible to all databases.
PG_LARGEOBJECT	Stores data making up large objects. A large object is identified by an OID assigned when it is created. Each large object is broken into segments or "pages" small enough to be conveniently stored as rows in pg_largeobject . The amount of data per page is defined as LOBLKSIZE . This system catalog is accessible only to system administrators.
PG_LARGEOBJECT_METADATA	Stores metadata associated with large objects. The actual large object data is stored in PG_LARGEOBJECT .
PG_NAMESPACE	Stores namespaces, that is, schema-related information.
PG_PARTITION	Stores all partitioned tables, table partitions, toast tables on table partitions, and index partitions in the database. Partitioned index information is not stored in the system catalog PG_PARTITION .
PG_PROC	Stores information about functions or procedures.

Common System Views (1)

System View Name	Description
GS_AUDITING	Displays all audit information about database-related operations. Only users with system administrator or security policy administrator permission can access this view.
GS_FILE_STAT	Records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.
GS_CLUSTER_RESOURCE_INFO	Displays a DN resource summary. This view can be queried only when enable_dynamic_workload is set to on and the view cannot be executed on DNs. Only users with sysadmin permission can query this view.
GS_LABELS	Displays all configured resource labels. Only users with system administrator or security policy administrator permission can access this view.
GS_MASKING	Displays all configured dynamic masking policies. Only users with system administrator or security policy administrator permission can access this view.
GS_MATVIEWS	Provides information about each materialized view in the database.
GS_SQL_COUNT	Displays statistics about five types of running statements (SELECT , INSERT , UPDATE , DELETE , and MERGE INTO) on the current node of the database.
GS_SESSION_TIME	Collects statistics about the running time of session threads and time consumed in each execution phase.
GS_SESSION_MEMORY	Collects statistics about memory usage at the session level, including all the memory allocated to Postgres and Stream threads on DNs for jobs currently executed by users.
GS_WLM_PLAN_OPERATOR_HISTO_RY	Displays the operator-level records of the execution plan after the job is executed on the primary node of the current user's database.
PG_LOCKS	Stores information about locks held by opened transactions.

Common System Views (2)

System View Name	Description
PG_ROLES	Provides information about database roles. Initialization users and users with the sysadmin or createrole attribute can view information about all roles. Other users can view only their own information.
PG_RULES	Provides access to query useful information about rewrite rules.
PG_TOTAL_USER_RESOURCE_INFO_OID	Displays resource usage of all users. Only administrators can query this view. This view is valid only if use_workload_manager is set to on .
PG_SETTINGS	Displays information about parameters of the running database.
PG_STATS	Provides access to the single-column statistics stored in the pg_statistic table.
PG_STAT_ACTIVITY	Displays information about the current user's queries.
PG_STAT_ALL_TABLES	Contains one row for each table in the current database (including TOAST tables), showing statistics about access to that specific table.
PG_TABLES	Provides useful information about access to each table in the database.
PG_TOTAL_USER_RESOURCE_INFO	Displays resource usage of all users. Only administrators can query this view. This view is valid only if use_workload_manager is set to on .
PG_USER	Provides information about database users. By default, only the initial user and users with the sysadmin attribute can view the information. Other users can view the information only after being granted permission.
PG_VIEWS	Provides useful information about access to each view in the database.
PLAN_TABLE	Displays plan information collected by EXPLAIN PLAN . Plan information is in a session-level lifecycle. After a session exits, the data will be deleted. Data is isolated between sessions and between users.

Reserved System Catalogs and System Views

- Reserved System Catalog
 - GS_WLM_USER_RESOURCE_HISTORY
- Reserved System Views
 - GS_WLM_OPERATOR_HISTORY
 - GS_WLM_OPERATOR_STATISTICS
 - GS_WLM_SESSION_HISTORY
 - GS_WLM_SESSION_INFO_ALL
 - GS_WLM_SESSION_STATISTICS
 - PG_SESSION_IOSTAT
 - PG_SESSION_WLMSTAT
 - PG_WLM_STATISTICS

Contents

1. Logical Structure
2. Storage Engine
3. Tablespace
4. User Roles
5. System Catalogs and System Views
- 6. Data Import and Export**
7. High-Risk Operations

Data Import and Export - INSERT

- Run the **INSERT** statement to add data into the openGauss database in either of the following ways:
 - Use the client tool provided by the openGauss database to add data to the openGauss database.
 - Connect to the database using the JDBC or ODBC driver and run the **INSERT** statement to add data into the openGauss database.
- You can add, modify, and delete database transactions for the openGauss database. **INSERT** is the simplest way to add data. It is applicable to scenarios with small data volume and low concurrency.

Data Import and Export – COPY FROM STDIN (1)

- Use either of the following methods to add data to openGauss using the **COPY FROM STDIN** statement:
 - Add data to openGauss by typing. COPY copies data between tables and files. COPY FROM copies data from a file to a table. COPY TO copies data from a table to a file.
 - COPY FROM and COPY TO apply to low concurrency and local data import and export in small amount.
 - Import data from a file or a database to openGauss through the CopyManager API of the JDBC driver. You can use any parameters in the COPY syntax.
 - CopyManager is an API class provided by the JDBC driver in openGauss. It is used to import data to the openGauss database in batches.
 - The CopyManager class is located in **org.postgresql.copy** package and inherits from the `java.lang.Object` class.

Data Import and Export – COPY FROM STDIN (2)

- COPY applies to the following scenarios:

- Import text data as small tables.
 - Export small tables.
 - Export a query result set.

- The usage is as follows:

- Import TEXT data.

```
copy t1 from '/data/input/t1.txt' delimiter '^';
```

- Export table data.

```
copy t1 to '/data/input/t1_output.txt' delimiter '^';
```

- Export a query result set.

```
copy (select * from t1 where a2=1) to '/data/input/t1_output.txt' delimiter '^';
```

Data Import and Export – gsql

- The gsql tool of openGauss provides the **\copy** meta-command to import and export data.
- **\copy** meta-command syntax

```
\copy { table [ ( column_list ) ] |  
       ( query ) } { from | to } { filename |  
       stdin | stdout | pstdin | pstdout }  
       [ with ] [ binary ] [ delimiter  
       [ as ] 'character' ] [ null [ as ] 'string' ]  
       [ csv [ header ] [ quote [ as ]  
       'character' ] [ escape [ as ] 'character' ]  
       [ force quote column_list | * ] [ force  
       not null column_list ] ]
```

- Notes:
 - You can run this command to import or export data after logging in to a database on any gsql client. Different from the COPY statement in SQL, this command performs read/write operations on local files rather than files on database servers. The accessibility and permissions **of local** files are restricted to local users.

Data Import and Export – `gs_dump`

- openGauss provides `gs_dump` and `gs_dumpall` to export required database objects and related information. You can use a tool to import the exported data to a target database for database migration.
- `gs_dump` exports a single database or its objects. `gs_dumpall` exports all databases or global objects in openGauss.

gs_dump Parameters

Parameter	Description
-U	Username for database connection.
-W	User password for database connection.
-f	Folder to store exported files.
-p	TCP port or local Unix-domain socket file name extension on which the server is listening for connections.
dbname	Name of the database to be exported.
-F	Format of the file to be exported. The values of -F are as follows: p : plaintext c : custom d : directory t : .tar
-n	Names of schemas to be exported. Data of the specified schemas will also be exported. Single schema: Enter -n schemaname . Multiple schemas: Enter -n schemaname for each schema.
-t	Table (or view, sequence, foreign table) to be exported. You can specify multiple tables by listing them or using wildcard characters. Single table: Enter -t schema.table . Multiple tables: Enter -t schema.table for each table.
-T	A list of tables, views, sequences, or foreign tables cannot be dumped. You can use multiple -T options or wildcard characters to specify tables.

Exporting a Database

- You can use `gs_dump` to export data and all object definitions of a database from openGauss. You can specify the information to export as follows:
 - Export full information of a database, including its data and all object definitions. You can use the exported information to create a duplicate database containing the same data as the current one.
 - Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a database that is the same as the current one, except the new database will not have data.
 - Export data of a database.
- Example: Use `gs_dump` to export the **postgres** database.

```
gs_dump -U jack -f /home/omm/backup/postgres_backup.tar -p 8000 postgres -F t
```

Exporting a Schema

- You can use gs_dump to export data and all object definitions of a schema from openGauss.
- You can export one or more specified schemas as needed. You can specify the information to export as follows:
 - Export full information of a schema, including its data and object definitions.
 - Export data of a schema, excluding its object definitions.
 - Export the object definitions of a schema, including the definitions of tables, stored procedures, and indexes.
- Example: Use gs_dump to export the **hr** and **public** schemas at the same time.

```
gs_dump -U jack -f /home/omm/backup/MPPDB_schema_backup -p 8000 human_resource -n hr -n public -F d
```

Exporting a Table

- You can use gs_dump to export data and definition of a table-level object from openGauss. Views, sequences, and foreign tables are special tables.
- You can export one or more specified tables as needed. You can specify the information to export as follows:
 - Export full information of a table, including its data and definition.
 - Export data of a table.
 - Export the definition of a table.
- Example: Use gs_dump to export the **hr.staffs** and **hr.employments** tables.

```
gs_dump -U jack -f /home/omm/backup/MPPDB_table_backup -p 8000 human_resource -t hr.staffs -t  
hr.employments -F d
```

Exporting all Databases

- You can use gs_dumpall to export full information of all databases in a cluster from openGauss, including information about each database and global objects in the cluster. You can specify the information to export as follows:
 - Export full information of all databases, including information about each database and global objects (such as roles and tablespaces) to openGauss. You can use the exported information to create a host environment containing the same databases, global objects, and data as the current one.
 - Export data of all databases, excluding all object definitions and global objects.
 - Export all object definitions of all databases, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a host environment that is the same as the current one, containing the same databases and tablespaces but no data.
- Example: Use gs_dumpall to export all database information at a time.

```
gs_dumpall -U omm -f /home/omm/backup/MPPDB_backup.sql -p 8000
```

Data Export By a User Without Required Permissions

- For `gs_dump` and `gs_dumpall`, use `-U` to specify the user that performs the export. If the specified user does not have the required permissions, data cannot be exported. In this case, you can set the `--role` parameter in the export command to specify the role that has the permission. Then, `gs_dump` or `gs_dumpall` uses the `--role` parameter to specify a role to export data.
- Example: Use `gs_dump` to export data of the **human_resource** database.
 - Assume that user **jack** does not have the permission to export data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set `--role` to **role1** in the export command.

```
gs_dump -U jack -f /home/omm/backup/MPPDB_backup.tar -p 8000 human_resource --role role1 --rolepassword abc@1234 -F t
```

Data Import and Export – gs_restore

- gs_restore is an import tool provided by the openGauss database. You can use this tool to import the files exported by gs_dump to a database. gs_restore can import the files in .tar, custom, or directory format.
- gs_restore can:
 - Import data to a database.
 - If a database is specified, data is imported to the database. For parallel import, the password for connecting to the database is required.
 - Import data to a script.
 - If no database is specified, a script containing the SQL statement to recreate the database is created and added to a file or standard output. This script output is equivalent to the plain text output of gs_dump.
- You can specify and sort the data to be imported.

gs_restore Parameters

Parameter	Description
-U	Username for database connection.
-W	User password for database connection.
-p	TCP port or local Unix-domain socket file name extension on which the server is listening for connections.
-d	Name of a database to which data will be imported.
-e	Exits if an error occurs when you send the SQL statement to the database. Error messages are displayed after the import process is complete.
-c	Cleans existing objects from the target database before the import.
-s	Imports only object definitions in schemas and does not import data. Sequence values will also not be imported.

gs_restore Example

- Run **gs_restore** to import all object definitions from the exported file of the whole **postgres** database to the **backupdb** database.

```
gs_restore -U jack /home/omm/backup/MPPDB_backup.tar -p 8000 -d backupdb -s -e -c
```

- Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.dmp** file (custom format).

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d backupdb
```

- Run **gs_restore** to import all object definitions of the **postgres** database to the **backupdb** database. Before the import, complete definitions and data exist in Postgres. After the import, all object definitions exist in the **backupdb** database and no data exists in the table.

```
gs_restore /home/omm/backup/MPPDB_backup.tar -p 8000 -d backupdb -s -e -c
```

ANALYZE

- The execution plan generator needs to use table statistics to generate the most effective query execution plan to improve query performance. After data is imported, you are advised to run the ANALYZE statement to update table statistics. The statistics are stored in the system catalog **PG_STATISTIC**.
- ANALYZE supports row-store and column-store tables. ANALYZE can also collect statistics about specified columns of a local table.
- ANALYZE syntax (the **product_info** table is used as an example)

```
postgres=# ANALYZE product_info;  
ANALYZE
```

VACUUM

- If a large number of rows were updated or deleted during import, run **VACUUM FULL** before **ANALYZE**. A large number of updates and deletions generate huge disk page fragments, which reduces query efficiency. VACUUM FULL can restore disk page fragments and return them to the OS.
- VACUUM FULL syntax (the **product_info** table is used as an example)

```
postgres=# VACUUM FULL product_info;  
VACUUM
```

Contents

1. Logical Structure
2. Storage Engine
3. Tablespace
4. User Roles
5. System Catalogs and System Views
6. Data Import and Export
- 7. High-Risk Operations**

Forbidden Operations

- The following lists forbidden operations during openGauss operation and maintenance.

Operation	Risk
Change the file name and permission in the data directory.	Serious errors occur on database nodes and cannot be fixed.
Delete database system catalogs or their data.	Service operations cannot be properly performed.

High-Risk Operations

- The following lists high-risk operations during openGauss operation and maintenance.

Category	Operation	Risk	Preventive Measure
Database	Open a configuration file and manually modify the port number.	The database fails to be started or connected.	Use the required tool to modify this file.
	Incautiously modify the content of the pg_hba.conf file.	The client fails to be connected.	Strictly follow the instructions provided in product documentation while you modify this file.
	Manually modify the pg_xlog file.	Database startup fails and data becomes inconsistent.	Use the required tool to modify this file.
Job	Run the kill -9 command to terminate a job process.	System resources occupied by this job cannot be released.	Log in to the database and use the pg_terminate_backend or pg_cancel_backend function to terminate the job, or press Ctrl+C to terminate the job process.

Quiz

1. (True or False) Users and roles have the same syntax in the openGauss database. Therefore, they are fully equivalent.
 - A. True
 - B. False
2. (Single-answer question) Which of the following commands can be used to export partial data from the student table?
 - A. gs_dump
 - B. gs_dumpall
 - C. copy *Conditional statement*
 - D. gs_backup

Quiz

3. (Multiple-answer question) Which of the following statements about row-store tables and column-store tables are true?
- A. Row-store tables are used in scenarios where INSERT and UPDATE operations are frequently performed.
 - B. Row-store tables are applicable to point queries, that is, simple index-based queries that return a small number of records.
 - C. Column-store tables are more efficient in compression and projection.
 - D. Row-store tables are used in scenarios where there are a large number of queries for statistical analysis.

Summary

- On completion of this course, you will be able to:
 - Understand the openGauss logical structure.
 - Master how to select different storage engines.
 - Master how to create and manage tablespaces.
 - Master how to manage users and roles.
 - Understand common system catalogs and views.
 - Master how to import and export data.
 - Understand high-risk operations of openGauss.

Recommendations

- HUAWEI CLOUD:
 - <https://www.huaweicloud.com/intl/en-us/>
- openGauss help document:
 - <https://opengauss.org/en/docs/2.0.0/docs/installation/installation.html>
- openGauss code repository:
 - <https://gitee.com/opengauss>
- GaussDB(for openGauss) help document:
 - https://support.huaweicloud.com/intl/en-us/productdesc-opengauss/opengauss_01_0002.html

Acronyms and Abbreviations

- MOT: memory-optimized table
- ACID: atomicity, consistency, isolation, and durability.
- NUMA: non-uniform memory access
- WAL: write-ahead logging
- TPC-C: A specification for online transaction processing (OLTP).

05 SQL Basics



Foreword

- openGauss is an open-source relational database management system. It supports SQL standards and is compatible with the SQL syntax as well as the capabilities of mainstream commercial databases.
- This chapter describes the SQL syntax, SQL syntax classification, common system functions, and operators, aiming to help beginners master the basic knowledge and operations of SQL.

Objectives

- Upon completion of this course, you will be able to:
 - Define SQL and classify SQL statements
 - List common data types and know how to use them
 - Understand the types and usage of common functions and operators

Contents

1. SQL Syntax Quick Start

- Introduction to SQL Statements
 - SQL Statement Classification
 - Data Type

2. SQL Syntax Classification

3. Operators and Common Functions

Introduction to SQL Statements

- Wikipedia:
 - Structured Query Language (SQL) is a domain-specific language used in programming. It is designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.
 - SQL is based on relational algebra and tuple relational calculus, including a data definition language (DDL) and data manipulation language (DML). The scope of SQL includes data insertion, query, update, and deletion, database schema creation and modification, and data access control.

Contents

1. SQL Syntax Quick Start

- Introduction to SQL Statements
- SQL Statement Classification
 - Data Type

2. SQL Syntax Classification

3. Operators and Common Functions

SQL Statement Classification

- DDL
 - Defines or modifies objects in a database, such as tables, indexes, views, databases, sequences, users, roles, tablespaces, and stored procedures.
- DML
 - Performs operations on data in database tables, such as inserting, updating, and deleting data.

SQL Statement Classification

- Data Control Language (DCL)
 - Sets or changes database transactions, performs authorization operations (such as granting permissions to users or roles, revoking permissions, creating roles, and deleting roles), locks tables (supporting shared and exclusive locks), and stops services.
- Data Query Language (DQL)
 - Queries data in a database, for example, querying data and combining the result sets of multiple SELECT statements.

Contents

1. SQL Syntax Quick Start

- Introduction to SQL Statements
- SQL Statement Classification
- Data Type

2. SQL Syntax Classification

3. Operators and Common Functions

Data Types

- A data type is a basic data attribute. Occupied storage space and allowed operations vary according to data types.
- Data in a database is stored in tables, and data types are defined for each column in the table. When storing data, you must comply with the attributes of these data types. Otherwise, errors may occur. The data types are as follows:
 - Common data types
 - Numeric, character, and date
 - Uncommon data types
 - Binary and Boolean

Numeric Types

Data Type	Description	Storage Space
Integers		
TINYINT	Tiny integer, also called INT1	One byte
SMALLINT	Small integer, also called INT2	Two bytes
INTEGER	Typical choice for integers, also called INT4	Four bytes
BINARY_INTEGER	Alias of the common integer type INTEGER	Four bytes
BIGINT	Big integer, also called INT8	Eight bytes
Arbitrary precision numbers		
NUMERIC[(p,s)], DECIMAL[(p,s)]	The value range of p (precision) is [1,1000], and the value range of s (scale) is [0, p]. p indicates the total digits, and s indicates the decimal digits.	Users specify precision. Two bytes are occupied for every four decimals of precision. An extra eight-byte overhead is added for numbers of this type.
NUMBER[(p,s)]	Alias of the NUMERIC type.	Users specify precision. Two bytes are occupied for every four decimals of precision. An extra eight-byte overhead is added for numbers of this type.

Example

- Create a table.

```
postgres=# CREATE TABLE type_t1
(
    a TINYINT,
    b INTEGER,
    c BIGINT,
    d DECIMAL(10,4)
);
```

- Add data.

```
postgres=# INSERT INTO type_t1 VALUES(100, 1000, 10000, 123456.1223);
```

- View data.

```
postgres=# SELECT * FROM type_t1;
```

- Delete the table.

```
postgres=# DROP TABLE type_t1;
```

Numeric Types

Data Type	Description	Storage Space
Serial integers		
SMALLSERIAL	Two-byte serial integer	Two bytes
SERIAL	Four-byte serial integer	Four bytes
BIGSERIAL	Eight-byte serial integer	Eight bytes
Floating-point numbers		
REAL, FLOAT4	Single-precision floating-point number, which is not very precise	Four bytes
DOUBLE PRECISION, FLOAT8	Double-precision floating-point number, which is not very precise	Eight bytes
FLOAT[(p)]	Floating-point number, which is not very precise. The value range of p (precision) is [1,53]. p is the precision, indicating the total decimal digits.	Four or eight bytes
BINARY_DOUBLE	Alias of DOUBLE PRECISION	Eight bytes
DEC[(p,s)]	The value range of p (precision) is [1,1000], and the value range of s (scale) is [0,p]. p indicates the total digits, and s indicates the decimal digits.	Users specify precision. Two bytes are occupied for every four decimals of precision. An extra eight-byte overhead is added for numbers of this type.
INTEGER[(p,s)]	The value range of p (precision) is [1,1000], and the value range of s (scale) is [0,p].	Users specify precision. Two bytes are occupied for every four decimals of precision. An extra eight-byte overhead is added for numbers of this type.

Example

- Create a table.

```
postgres=# CREATE TABLE type_t2
(
    a SMALLSERIAL,
    b SERIAL,
    c BIGSERIAL,
    d FLOAT4,
    e DECIMAL(10,4)
);
```

- Add data.

```
postgres=# INSERT INTO type_t2 VALUES(default, default, default, 10.365456, 123.123654);
postgres=# INSERT INTO type_t2 VALUES(default, default, default, 10.365456, 123.123654);
```

- View data.

```
postgres=# SELECT * FROM type_t2;
```

- Delete the table.

```
postgres=# DROP TABLE type_t2;
```

Character Types

Data Type	Description	Storage Space
Character type		
CHAR(n) CHARACTER(n) NCHAR(n)	Character string with fixed length. Empty characters are filled in with blank spaces. n indicates the string length. If it is not specified, the default precision of 1 is used.	Maximum: 10 MB
VARCHAR(n) CHARACTER VARYING(n)	Character string with variable length. n indicates the string length.	Maximum: 10 MB
VARCHAR2(n)	Character string with variable length. It is the alias of the VARCHAR(n) type. n indicates the string length.	Maximum: 10 MB
NVARCHAR2(n)	Character string with variable length. n indicates the string length.	Maximum: 10 MB
TEXT	Character string with variable length.	Maximum: 1 GB minus 1 byte
CLOB	Big text object. It is the alias of the TEXT type.	Maximum: 1 GB minus 1 byte

Example

- Create a table.

```
postgres=# CREATE TABLE type_t3
(
    a CHARACTER(4),
    b VARCHAR(5),
);
```

- Add data.

```
postgres=# INSERT INTO type_t3 VALUES('ok', 'good');
```

- View data.

```
postgres=# SELECT * FROM type_t3;
```

- Delete the table.

```
postgres=# DROP TABLE type_t3;
```

Date Types

Data Type	Description	Storage Space
Date/Time types		
DATE	Date and time.	Four bytes
TIME [(p)] [WITHOUT TIME ZONE]	Time within one day. p indicates the precision after the decimal point. The value ranges from 0 to 6.	Eight bytes
TIME [(p)] [WITH TIME ZONE]	Time within one day (with time zone). p indicates the precision after the decimal point. The value ranges from 0 to 6.	12 bytes
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	Date and time. p indicates the precision after the decimal point. The value ranges from 0 to 6.	Eight bytes
TIMESTAMP[(p)][WITH TIME ZONE]	Date and time (with time zone). TIMESTAMP is also called TIMESTAMPTZ. p indicates the precision after the decimal point. The value ranges from 0 to 6.	Eight bytes
SMALLDATETIME	Date and time (without time zone). The precision is minute. A duration between 30s and 60s is rounded to 1 minute.	Eight bytes
INTERVAL DAY (I) TO SECOND (p)	Time interval (X days X hours X minutes X seconds). I indicates the precision of days. The value ranges from 0 to 6. p indicates the precision of seconds. The value ranges from 0 to 6. The digit 0 at the end of a decimal number is not displayed.	16 bytes
INTERVAL [FIELDS] [(p)]	Time interval. The value of FIELDS can be YEAR , MONTH , DAY , HOUR , MINUTE , SECOND , or DAY TO HOUR . p indicates the precision of seconds. The value ranges from 0 to 6. p takes effect only when the value of FIELDS is SECOND , DAY TO SECOND , HOUR TO SECOND , or MINUTE TO SECOND .	12 bytes

Example

- Create a table.

```
postgres=# CREATE TABLE type_t4
(
    a TIME WITHOUT TIME ZONE,
    b TIME WITH TIME ZONE,
    c TIMESTAMP WITHOUT TIME ZONE,
    d TIMESTAMP WITH TIME ZONE,
    e SMALLDATETIME
);
```

- Add data.

```
postgres=# INSERT INTO type_t4 VALUES('21:21:21','21:21:21 pst','2010-12-12','2013-12-11 pst','2003-04-12 04:05:06');
```

- View data.

```
postgres=# SELECT * FROM type_t4;
      a      b      c          d          e
-----+
21:21:21 21:21:21-08 2010-12-12 00:00:00 2013-12-11 16:00:00+08 2003-04-12 04:05:00
```

- Delete the table.

```
postgres=# DROP TABLE type_t4;
```

Uncommon Data Types

- Binary

Data Type	Description	Occupied Space
BYTEA	Variable-length binary string.	Four bytes plus the actual binary string. The maximum size is 1073733621 bytes (1 GB minus 8203 bytes).
RAW	Variable-length hexadecimal string.	Four bytes plus the actual hexadecimal string. The maximum size is 1 GB minus 8203 bytes.
BYTEAWITHOUTORDERWITHEQUALCOL	Variable-length binary character string (new type for encryption). If deterministic encryption is specified in the encrypted column, the column type is BYTEAWITHOUTORDERWITHQUALCO.	Four bytes plus the actual binary string. The maximum size is 1073741771 bytes (1 GB minus 53 bytes).
BLOB	Stores binary data of variable-length objects. It is the large object data of the raw type.	The maximum size is 1 GB minus 8203 bytes.

- Boolean

Data Type	Description	Value Range	Occupied Space
BOOLEAN	Stores Boolean data.	TRUE,FALSE	One byte

Example

- Create a table.

```
postgres=# CREATE TABLE type_t5
(
    a BLOB,
    b RAW,
    c BYTEA,
    e BOOLEAN
);
```

- Add data.

```
postgres=# INSERT INTO type_t5 VALUES(empty_blob(), HEXTORAW('DEADBEEF'),E'\\xDEADBEEF', TRUE);
```

- View data.

```
postgres=# SELECT * FROM type_t5;
   a      b      c      d
-----+
DEADBEEF  \xdeadbeef  t
```

- Delete the table.

```
postgres=# DROP TABLE type_t5;
```

Quiz

1. (True or false) BIGINT occupies four bytes.
 - A. True
 - B. False
2. (True or false) CLOB is used to store large object data of the TEXT type.
 - A. True
 - B. False

Contents

1. SQL Syntax Quick Start

2. SQL Syntax Classification

- Data Query

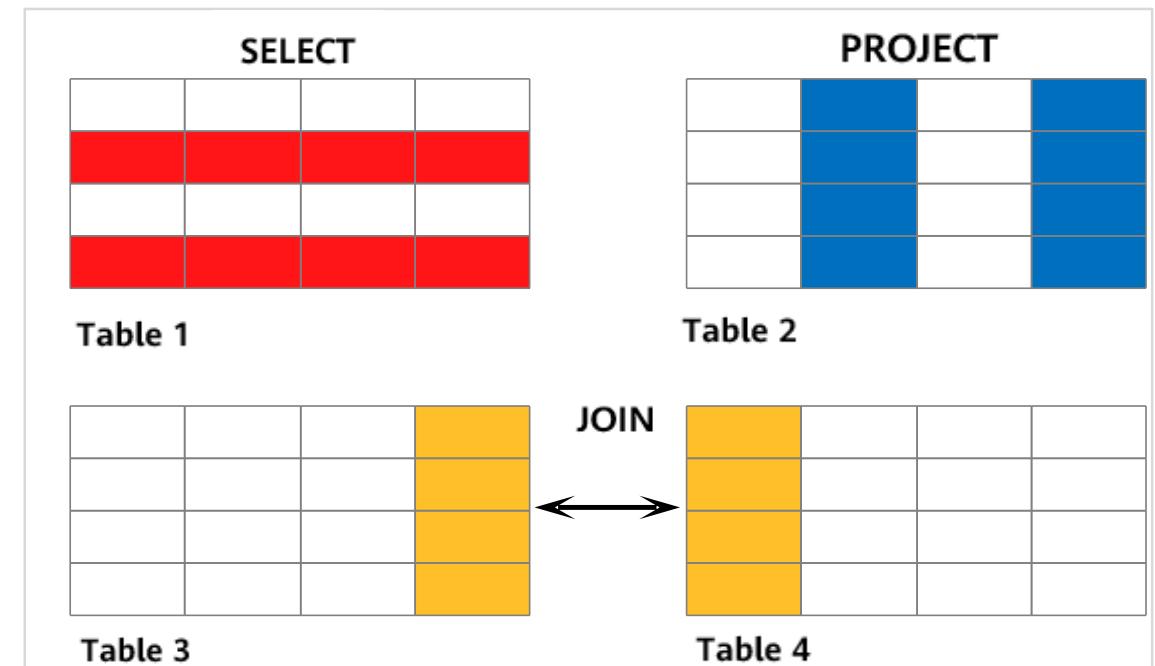
- Data Update

- Data Definition

3. Operators and Common Functions

Query Overview

- Data in a database is stored in tables. Generally, these tables consist of tuples and columns. Columns form the structure of a table and rows form the data of a table.
- The SELECT statement can be used to query required information from a database table, but does not change the said data. All it does is to extract and display the data.
- Key elements of the SELECT statement
 - **SELECT:** Queries the tuples that meet the conditions and filters out those that do not.
 - **PROJECT:** Queries the columns that meet the conditions and filters out those that do not.
 - **JOIN:** Queries data from multiple tables. Joins are the core of a relational database. Data is stored in different tables, and complete information is obtained through queries using **JOIN**.



Simple Query

- Syntax:

```
SELECT /*+ plan_hint */ [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM FROM_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST | LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ , ... ]
```

- Usage:

- The expression between the **SELECT** keyword and the **FROM** clause is called a **SELECT** item. **SELECT** items are used to specify the columns to be queried, and the **FROM** clause specifies the table where the columns are located.

Example

- Use an asterisk (*) after **SELECT** to query all columns in the **bank_card** table.

```
SELECT * FROM bank_card;
b_number          | b_type      | b_c_id
-----+-----+
6222021302020000001 | Credit Card | 1
6222021302020000002 | Credit Card | 3
6222021302020000003 | Credit Card | 5
```

Selecting a Query Column

- When you select a query column, the column name can be expressed in the following ways:
 - Manually enter column names and separate them with commas (,).

```
SELECT a, b, f1, f2 FROM t1, t2;
```

The columns **a** and **b** are columns in table **t1**, and **f1** and **f2** are columns in table **t2**.

- Use calculated columns.

```
SELECT a+b FROM t1;
```

- If two or more tables have the same column names, you are advised to specify both table and column names. Query results can be returned without specifying column names but will require extra work. Example:

```
SELECT t1.f1, t2.f1 FROM t1, t2;
```

Example

- View the card number and card type in the **bank_card** table.

```
postgres=# SELECT b_number,b_type FROM bank_card;
b_number          |      b_type
-----+-----
6222021302020000001 | Credit Card
6222021302020000002 | Credit Card
6222021302020000003 | Credit Card
```

- View the ID, name, email address, and bank card number of customer 1.

```
postgres=# SELECT a.c_id,a.c_name, a.c_mail, b.b_number FROM client a, bank_card b where a.c_id= 1 and
b.b_c_id = 1;

c_id      | c_name    |   c_mail           |   b_number
-----+-----+-----+-----+
1       | Zhang Yi | zhangyi@Huawei.com | 6222021302020000001
```

Alias

- You can use the **AS some_name** clause to specify an alias for a table or column. Generally, an alias is created to improve the readability of a table or column name.
- Syntax
 - The SQL aliases of columns and tables follow the corresponding column and table names. You can choose whether to add the keyword **AS** in the middle.
- Example: Use an alias.

```
postgres=# SELECT b_c_id AS CardID, b_type CardType FROM bank_card;  
cardid | cardtype  
-----+  
1     | Credit Card  
3     | Credit Card  
5     | Savings Card
```

```
postgres=# SELECT a.c_id CID ,a.c_name Name, a.c_mail Email, b.b_number "Card Number" FROM client a, bank_card b where  
a.c_id= 1 and b.b_c_id = 1;
```

```
cid | name | email | Card Number  
---+---+---+  
1  | Zhang Yi | zhangyi@Huawei.com | 6222021302020000001
```

Conditional Query

- In a SELECT statement, you can specify conditions for a more accurate query. Conditions are specified by using expressions and operators, and the return value is **TRUE**, **FALSE**, or **UNKNOWN**. Query conditions are used in the **WHERE** and **HAVING** clauses.
- Syntax:

- **condition clause**

```
select_statement { predicate } [ { AND | OR } condition ] [ , ... n ]
```

- **predicate clause**

```
{ expression { = | <> | != | > | >= | < | <= } { ALL | ANY } expression | ( SELECT )
| string_expression [ NOT ] LIKE string_expression
| expression [ NOT ] BETWEEN expression AND expression
| expression IS [ NOT ] NULL
| expression [ NOT ] IN ( SELECT | expression [ , ... n ] )
| [ NOT ] EXISTS ( SELECT )
}
```

Conditional Query

- Query conditions are defined by using expressions and operators. Common conditions are defined as follows:
 - Comparison operators (such as `>`, `<`, `>=`, `<=`, `!=`, `<>`, and `=`), specify the comparison query conditions. In the expression, single quotation marks (`'`) are optional for the numeric type but mandatory for character and date types.
 - Test operators specify query ranges. To specify multiple conditions, use the **AND** logical operator to connect these conditions.
To specify one of multiple conditions, use the **OR** logical operator to connect these conditions.
- Example: Use a comparison operator to query the information about a credit card.

```
postgres=# SELECT * FROM bank_card WHERE b_type='Credit Card';
b_number          | b_type      | b_c_id
-----+-----+-----+
6222021302020000001 | Credit Card | 1
6222021302020000002 | Credit Card | 3
6222021302020000003 | Credit Card | 5
6222021302020000004 | Credit Card | 7
6222021302020000005 | Credit Card | 9
```

Conditional Query

- Logical Operators
 - Common logical operators include AND, OR, and NOT. The operation result can be **TRUE**, **FALSE**, or **NULL** (which means unknown). Their priorities are NOT > AND > OR.

a	b	a AND b	a OR b	NOT a
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

- Example: Query information about a credit card whose customer ID is 1 from the **bank_card** table.

```
postgres=# SELECT* FROM bank_card where b_c_id= 1and b_type='Credit Card';
          b_number      |   b_type      |   b_c_id
-----+-----+-----+
 6222021302020000001 | Credit Card | 1
```

Join Query

- Generally, data to be queried is distributed in multiple tables. A query across tables or views is called a join query. In most cases, a join query is executed between a table and its child tables.
- Syntax

```
SELECT [ , ... ] FROM table_reference  
      [LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER] | INNER]  
      JOIN table_reference  
      [ON { predicate } [ { AND | OR } condition ] [ , ... n ]]
```

- **table_reference** clause

```
{ table_name [ [AS] alias ]  
| view_name [ [AS] alias ]  
| ( SELECTquery ) [ [AS] alias ]  
}
```

Join Query

- If multiple tables are included in the **FROM** clause for a query, the database executes a join query.
 - The **SELECT** items of the query can be any columns in these tables.

```
SELECT table1.column, table2.column FROM table1, table2;
```

- Most join queries contain at least one query condition, which can be either in the **FROM** or **WHERE** clause.

```
SELECT table1.column, table2.column FROM table1 JOIN table2 ON(table1.column1 = table2.column2);  
SELECT table1.column, table2.column FROM table1, table2 WHERE table1.column1 = table2.column2;
```

- In the **WHERE** clause, you can use the operator (+) to convert a table join to an outer join. However, this method is not recommended because it is not the standard SQL syntax.

Inner Join

- **INNER JOIN:** The syntax keyword is **INNER JOIN**, where **INNER** can be omitted. If an inner join is used, the join execution sequence will follow the sequence of tables in the statement.
- Example: Query the customer ID, bank card number, and bank card type. Use the **c_id** column of the **client** and **bank_card** tables to perform the query operation.

```
postgres=# SELECT c.c_id, b.b_number, b.b_type FROM client c JOIN bank_card b ON (b.b_c_id = c.c_id);
c_id      |      b_number          |      b_type
-----+-----+-----+
 1      | 6222021302020000001  | Credit Card
 3      | 6222021302020000002  | Credit Card
 5      | 6222021302020000003  | Credit Card
 7      | 6222021302020000004  | Credit Card
 9      | 6222021302020000005  | Credit Card
 10     | 6222021302020000006  | Credit Card
 12     | 6222021302020000007  | Credit Card
 14     | 6222021302020000008  | Credit Card
```

Outer Join

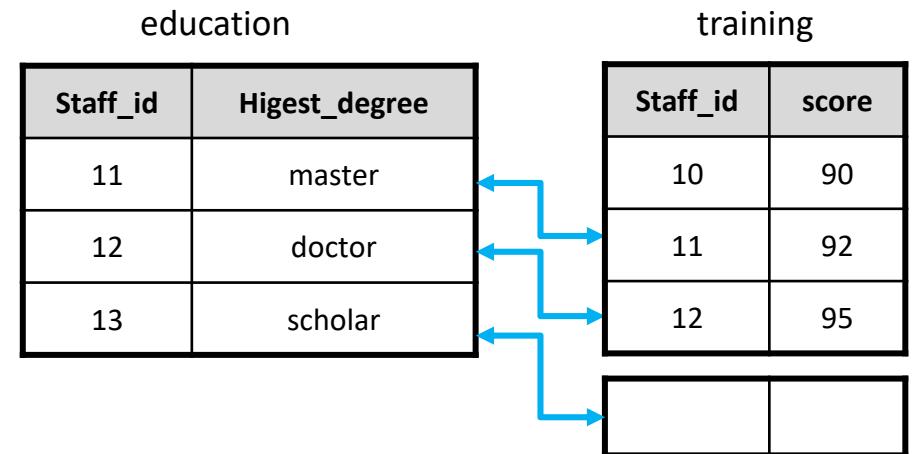
- In inner joins, the two data sources specified are equal. However, in outer joins, one data source is used as the base for the other source to match according to conditions.
- An inner join returns all data records in both tables that meet join conditions. An outer join returns both records that meet join conditions and those that do not.
- Outer joins include left, right, and full outer joins.

Left Outer Join

- Also called a left join, indicating that the query is driven by the left table.
- All data from the left table is combined with data from the right table which satisfies the specified join conditions. Columns of the right table that do not match the conditions are set to **NULL**.
- Example:

```
postgres=# SELECT e.staff_id, e.higest_degree, t.score FROM education e LEFT JOIN training t ON (e.staff_id = t.staff_id);
```

STAFF_ID	HIGEST_DEGREE	SCORE
11	master	92
12	doctor	95
13	scholar	



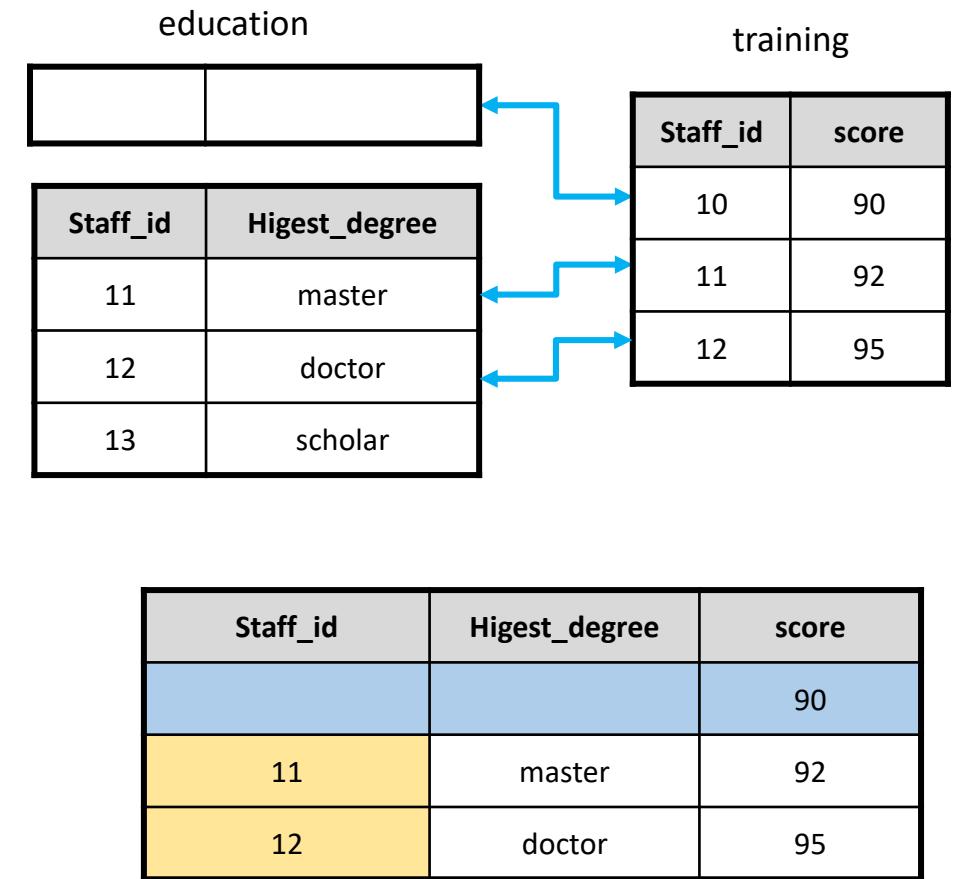
Staff_id	Higest_degree	score
11	master	92
12	doctor	95
13	scholar	

Right Outer Join

- Also called a right join, indicating that the query is driven by the right table.
- All data from the right table is combined with data from the left table which satisfies the specified join conditions. Columns of the left table that do not match the conditions are set to **NULL**.
- Example:

```
postgres=# SELECT e.staff_id, e.higest_degree, t.score FROM education e RIGHT JOIN training t ON (e.staff_id = t.staff_id);
```

STAFF_ID	HIGEST_DEGREE	SCORE
		90
11	master	92
12	doctor	95

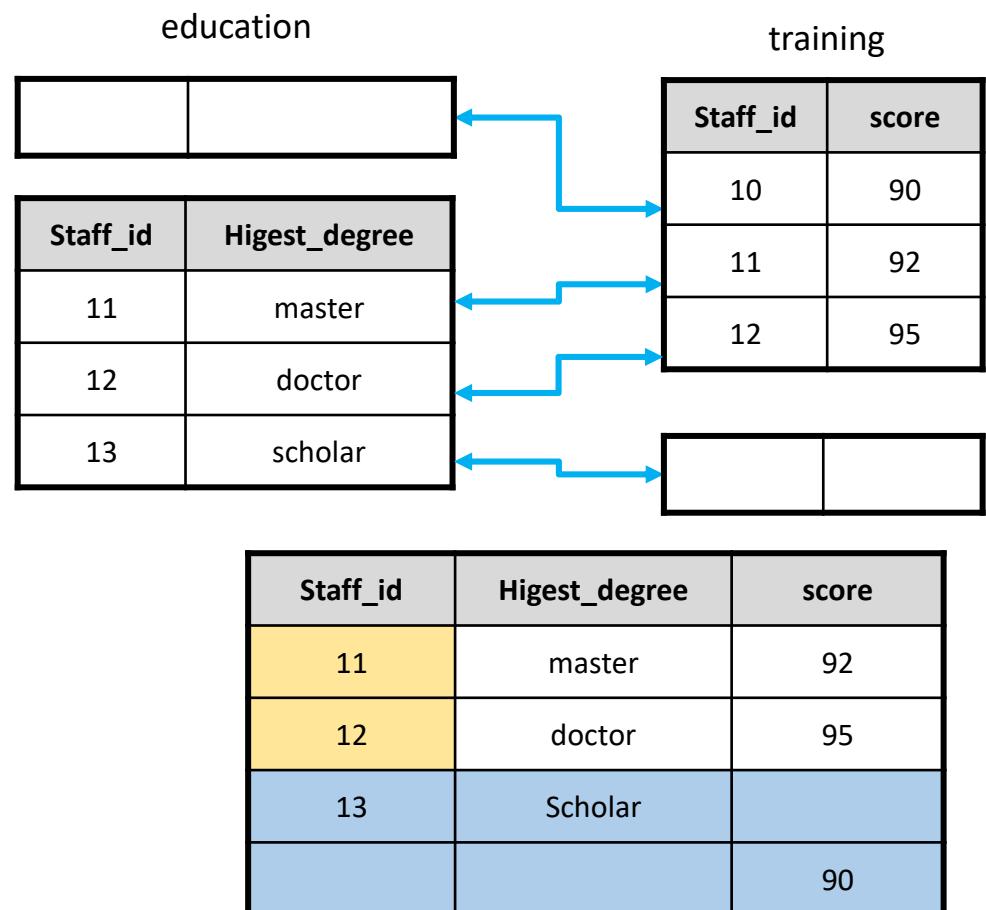


Full Outer Join

- Also called a full join; the query returns all records (whether or not they satisfy the conditions) from either the left or right table. The other side that does not meet join conditions is filled with **NULL**.
- Example:

```
postgres=# SELECT e.staff_id, e.higest_degree, t.score FROM education e FULL JOIN training t ON (e.staff_id = t.staff_id);
```

STAFF_ID	HIGEST_DEGREE	SCORE
11	master	92
12	doctor	95
13	scholar	90



Data Sorting

- **ORDER BY** clause
 - The **ORDER BY** clause is used to sort rows returned by a query by a specified column. If there is no **ORDER BY**, the same query may retrieve data organized in different orders.
- Syntax

```
ORDER BY { column_name | number | expression } [ ASC | DESC ][ NULLS FIRST | NULLS LAST ] [ , ... ]
```

- Usage
 - By default, the **ORDER BY** clause sorts records in ascending order. For descending order, use the keyword **DESC**.
 - The **NULLS FIRST | NULLS LAST** keyword specifies the position of **NULL** values in the **ORDER BY** column. **FIRST** indicates that **NULL** values are placed before non-**NULL** values and **LAST** indicates that **NULL** values are placed after non-**NULL** values. If this keyword is not specified, **NULLS FIRST** is the default for **ASC** and **NULLS LAST** is the default for **DESC**.

Example

- Query the names, amounts, and target people of insurances with insurance numbers higher than 2 in descending order.

```
postgres=# SELECT i_name,i_amount,i_person FROM insurance WHERE i_id>2 ORDER BY i_amount DESC;  
i_name              |   i_amount    |   i_person  
-----+-----+-----  
Accident Insurance  |   5000        |   all  
Medical Insurance  |   2000        |   all  
Loss of Property Insurance  |   1500        |   Middle-aged
```

Data Limit

- The data limit function involves two independent clauses: **LIMIT** and **OFFSET**.
 - The **LIMIT** clause allows users to limit the rows returned by a query.

```
LIMIT { count | ALL }
```

- The **OFFSET** clause sets the position from which the return begins.

```
OFFSET start
```

- Usage
 - **start**: Specifies the number of rows to be skipped before returning rows.
 - **count**: Specifies the maximum number of rows to be returned.
 - When both **start** and **count** are specified, the *start* rows are skipped before the *count* rows to be returned are calculated.
 - **LIMIT 5,20** is equivalent to **LIMIT 20 OFFSET 5** and **OFFSET 5 LIMIT 20**.

Example

- Query the insurance information in the following table. Add **LIMIT 2 OFFSET 1** to skip the first row and query a total of two rows.

```
postgres=# SELECT i_name, i_id, i_amount, i_person FROM insurance LIMIT 2 OFFSET 1;
```

i_name	i_id	i_amount	i_person
Life Insurance	2	3000	Seniors
Accident Insurance	3	5000	All

Quiz

1. (Single-choice) Which of the following logical expressions is used to search for records whose position is an engineer and salary is greater than 6000?
 - A. Position = 'Engineer' OR Salary > 6000
 - B. Position = Engineer AND Salary > 6000
 - C. Position = Engineer OR Salary > 6000
 - D. Position = 'Engineer' AND Salary > 6000
2. (Single-choice) In a **WHERE** clause, the expression "age between 20 and 30" is equivalent to.
 - A. age >= 20 and age <= 30
 - B. age >= 20 or age <=30
 - C. age > 20 and age < 30
 - D. age > 20 and age < 30

Quiz

3. (Multiple-choice) Assume that you want to query student names, ages, and scores from the **student** table, and want the results sorted in descending order by age. If there are students of the same age, they are sorted in ascending order by score instead. Which of the following SQL statements can be used?
- A. SELECT name, age, score FROM student order by age desc, score;
 - B. SELECT name, age, score FROM student order by age, score asc;
 - C. SELECT name, age, score FROM student order by 2 desc, 3 asc;
 - D. SELECT name, age, score FROM student order by 1 desc, 2;

Contents

1. SQL Syntax Quick Start

2. SQL Syntax Classification

- Data Query

- Data Update

- Data Definition

3. Operators and Common Functions

Inserting Data

- Description
 - **INSERT** inserts data into a table.
- Precautions
 - You must have the **INSERT** permission for a table in order to insert into it.
 - You must have the **SELECT** permission for the table in order to use the **RETURNING** clause.
 - If **ON DUPLICATE KEY UPDATE** is used, you must have the **SELECT** and **UPDATE** permissions for the table and the **SELECT** permission for the unique constraint (primary key or unique index).
 - If you use the **query** clause to insert rows from a query, you need to have the **SELECT** permission for any table or column used in the query.

Inserting Data

- Syntax
 - There are three types of the **INSERT** statements.
 - Insert values. Create a row of records, and insert them into a table.

```
INSERT [hint_info] [IGNORE] [ INTO ] [ schema_name. ]table_name [ ( column_name [ , ... ] ) ] VALUES ( expression [ , ... ] )
```
 - Insert a query. Create one or more rows of records based on the result set returned by **SELECT** and insert the records into a table.

```
INSERT [IGNORE] [ INTO ] [ schema_name. ]table_name [table_alais][ ( column_name [ , ... ] ) ]select_clause
```
 - Insert records into a table. If primary key conflicts are reported, values in the specified column are updated.

```
INSERT [ INTO ] [ schema_name. ]table_name [ ( column_name [ , ... ] ) ] VALUES ( expression [ , ... ] ) ON  
DUPLICATE KEY UPDATE {column_name = expression} [ , ... ]
```

Example

- Insert data into the **bank_card** table.

- Create a table named **bank_card**.

```
postgres=# CREATE TABLE bank_card(b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20), b_c_id INT NOT NULL);
```

- Insert a value. Insert a record into the **bank_card** table.

```
postgres=# INSERT INTO bank_card(b_number,b_type,b_c_id) VALUES ('6222021302020000001', 'Credit Card',1);
```

- Insert all data from the **bank_card** table into the **bank_card1** table using a subquery.

```
postgres=# INSERT INTO bank_card1 SELECT * FROM bank_card;
```

- Insert a record and update a value for columns in which this primary key is a duplicate.

```
INSERT INTO bank_card VALUES ('6222021302020000001', 'Credit Card',1) ON DUPLICATE KEY UPDATE b_type = 'Savings Card';
```

Updating Data

- Description
 - **UPDATE** updates row values in a table.
- Precautions:
 - You must have the **UPDATE** permission for tables to be updated.
 - You must have the **SELECT** permission for all tables involved in the expressions or conditions.
 - The **RETURNING** clause is currently not supported for column-store tables.
 - Column-store tables cannot be updated if the results are uncertain. If you update one row of data with multiple rows of data in a column-store table, an error will be reported.
 - Memory that holds old records is not recycled after a column-store table is updated. You need to clean it by executing **VACUUM FULL table_name**.
 - **UPDATE** is currently not supported for column-store replication tables.

Updating Data

- Syntax

```
UPDATE table_reference SET { [col_name = expression] [ , ... ] | (col_name[,...]) = (SELECT expression[,...]) } [ WHERE condition ]
```

- **table_reference clause**

```
{ [ schema_name. ] table_name  
| join_table  
}
```

- **join_table clause**

```
table_reference [LEFT [OUTER] | RIGHT [OUTER] | INNER ] JOIN table_reference ON conditional_expr
```

(col_name[,...]) = (expression[,...]) can be used only when the **join_table** clause is used.

- Example: Update the record whose customer ID is 1 in the **bank_card** table and change the value of **b_type** to **Credit Card**.

```
postgres=# UPDATE bank_card SET b_type = 'Credit Card' WHERE b_c_id=1;
```

Deleting Data

- Description
 - **DELETE** deletes records from a table.
- Precautions
 - You must have the **DELETE** permission for the table.
 - You must have the **SELECT** permission for any table in the **USING** clause or whose values are read in **condition**.
 - The **RETURNING** clause is currently not supported for column-store tables.

Deleting Data

- **Syntax**

```
DELETE FROM [ schema_name. ]table_name  
[ WHERE condition ]  
[ ORDER BY { column_name [ ASC | DESC ] [ NULLS FIRST | NULLS LAST ] } [ , ... ] ]  
[ LIMIT [ start, ] count  
| LIMIT count OFFSET start  
| OFFSET start[ LIMIT count ] ]
```

Delete the records from a table if they match the records in another table.

```
DELETE table_ref_list FROM join_table
```

Or

```
DELETE FROM table_ref_list USING join_table
```

Example

- Delete the records, whose **b_c_id** is **10** and whose bank card type is credit card, from the **bank_card** table.
 - Delete the **bank_card** table.

- ```
postgres=# DROP TABLE IF EXISTS bank_card;
```

- ```
postgres=# CREATE TABLE bank_card( b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20),b_c_id INT NOT NULL);
```

- ```
postgres=# INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222021302020000021','Savings Card', 30);
postgres=# INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222021302020000022','Savings Card', 31);
postgres=# INSERT INTO bank_card(b_number, b_type, b_c_id) VALUES ('6222021302020000023','Savings Card', 32);
```

- 

- ```
postgres=# DELETE FROM bank_card WHERE b_type='Credit Card' AND b_c_id=10;
```

Quiz

1. (Single-choice) Which of the following SQL commands can be used to increase the value of the **AGE** column in the **STAFFS** table by five years?
 - A. UPDATE SET AGE WITH AGE+5
 - B. UPDATE AGE WITH AGE+5
 - C. UPDATE STAFFS SET AGE = AGE+5
 - D. UPDATE STAFFS AGE WITH AGE+5
2. (Single-choice) Which of the following four groups of SQL commands belong to the DML commands?
 - A. CREATE, DROP, UPDATE
 - B. INSERT, UPDATE, DELETE
 - C. INSERT, DROP, ALTER
 - D. UPDATE, DELETE, ALTER

Quiz

3. (Single-choice) Which of the following SQL statements will delete records for all students whose class ID (specified by **cid**) is 6 from the **student** table?
- A. delete FROM student where cid = 6;
 - B. delete * FROM student where cid = 6;
 - C. delete FROM student on cid = 6;
 - D. delete * FROM student on cid = 6;

Contents

1. SQL Syntax Quick Start

2. SQL Syntax Classification

- Data Query

- Data Update

- Data Definition

3. Operators and Common Functions

DDL Classification

- Database objects are part of what makes up a database. They include tables, indexes, views, stored procedures, default values, rules, triggers, users, and functions.
- Three DDL statements are used to define, modify, and remove objects in a database:
 - **CREATE** is used to create database objects.
 - **ALTER** is used to modify the attributes of database objects.
 - **DROP** is used to delete database objects.

Defining a Database

- A database is the warehouse for organizing, storing, and managing data. Defining a database includes creating a database, modifying database attributes, and deleting a database.

Function	SQL Statement
Creating a database	CREATE DATABASE
Modifying database attributes	ALTER DATABASE
Deleting a database	DROP DATABASE

Creating a Database

- Syntax

```
CREATE DATABASE database_name
  [ [ WITH ] { [ OWNER [=] user_name ] |
    [ TEMPLATE [=] template ] |
    [ ENCODING [=] encoding ] |
    [ LC_COLLATE [=] lc_collate ] |
    [ LC_CTYPE [=] lc_ctype ] |
    [ DBCOMPATIBILITY [=] compatibility_type ] |
    [ TABLESPACE [=] tablespace_name ] |
    [ CONNECTION LIMIT [=] connlimit ] }[...] ];
```

- Note

- Only system administrators or users with the **CREATEDB** permission can create a database.
- **CREATE DATABASE** cannot be executed within a transaction block.
- If an error message similar to "could not initialize database directory" is displayed during database creation, permissions or disk space for the file system directory may be insufficient.

Example

- Create a database user named **jim**.

```
postgres=# CREATE USER jim PASSWORD 'Bigdata@123';
postgres=# CREATE USER jim01 PASSWORD 'Bigdata@123';
```

- Create a tablespace.

```
postgres=# CREATE TABLESPACE tablespace01 RELATIVE LOCATION 'tablespace01/tablespace01';
```

- Create database **music** using **template0** and specify user **jim** as its owner.

```
postgres=# CREATE DATABASE music OWNER jim TEMPLATE template0;
```

Modifying a Database

- **ALTER DATABASE** modifies a database, including its name, owner, connection limitation, and object isolation.
- Syntax

```
ALTER DATABASE database_name
  [ [ WITH ] [ CONNECTION LIMIT connlimit ] ]
  [ RENAME TO new_name ]
  [ OWNER TO new_owner ]
  [ SET TABALSPACE new_tablespace]
  [...];
```

- Note
 - To modify the name of a database, you must have the **CREATEDB** permission.
 - To modify the owner of a database, you must be the database owner or system administrator and a member of the new owner role, with the **CREATEDB** permission.
 - To modify the default tablespace of a database, you must have the **CREATE** permission to create a tablespace. This statement physically migrates tables and indexes in a default tablespace to a new tablespace. Note that tables and indexes outside the default tablespace are not affected.

Example

- Change the maximum number of connections to database **music**.

```
postgres=# ALTER DATABASE music WITH CONNECTION LIMIT 30;
```

- Rename the database.

```
postgres=# ALTER DATABASE music RENAME TO music01;
```

- Change the default tablespace of the database.

```
postgres=# ALTER DATABASE music01 SET TABLESPACE tablespace01;
```

- Change the database owner.

```
postgres=# ALTER DATABASE music01 OWNER TO jim01;
```

Deleting a Database

- Syntax

```
DROP DATABASE [ IF EXISTS ] database_name;
```

- Example:

- Delete database **music**.

```
DROP DATABASE music01;
```

- Delete a user.

```
DROP USER jim;  
DROP USER jim01;
```

- Delete a tablespace.

```
DROP TABLESPACE tablespace01;
```

Defining a Schema

- A schema is a set of database objects and is used to control the access to the database objects.

Function	SQL Statement
Creating a schema	CREATE SCHEMA
Modifying schema attributes	ALTER SCHEMA
Deleting a schema	DROP SCHEMA

Creating a Schema

- Syntax
 - Create a schema based on a specified name.

```
CREATE SCHEMA schema_name  
[ AUTHORIZATION user_name ] [ schema_element [ ... ] ];
```

- Create a schema based on a username.

```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```

- Note
 - Only a user with the **CREATE** permission for the current database can perform this operation.
 - The owner of an object created by the system administrator in a schema with the same name as a common user is that common user, not the system administrator.

Example

- Create a role named **role1**.

```
postgres=# CREATE ROLE role1 IDENTIFIED BY 'Bigdata@123';
```

- Create a schema named **role1** for the **role1** role. The owner of the **films** and **winners** tables is **role1**, as they are created by using sub-commands under the **role1** schema.

```
postgres=# CREATE SCHEMA AUTHORIZATION role1
CREATE TABLE films (title text, release date, awards text[])
CREATE VIEW winners AS
SELECT title, release FROM films WHERE awards IS NOT NULL;
```

Modifying a Schema

- Modifying schema attributes

- Rename the schema.

```
postgres=# ALTER SCHEMA role1 RENAME TO role1_new;
```

- Create a user named **jack**.

```
postgres=# CREATE USER jack PASSWORD 'Bigdata@123';
```

- Change the owner of the schema.

```
postgres=# ALTER SCHEMA role1_new OWNER TO jack;
```

- Note

- Only the owner of a schema or the system administrator has the permission to run the **ALTER SCHEMA** statement.

Deleting a Schema

- Deleting a schema from a database
- Syntax

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

- Example: Delete user **jack** and schema **role1_new**.

```
postgres=# DROP SCHEMA role1_new;  
postgres=# DROP USER jack;
```

- Note

- Only the owner of a schema or the system administrator has the **DROP SCHEMA** permission.

Defining a Table

- A table is a special data structure in a database and is used to store data objects and relationships between data objects. The following table lists the involved SQL statements.

Function	SQL Statement
Creating a table	CREATE TABLE
Modifying table attributes	ALTER TABLE
Deleting a table	DROP TABLE
Deleting all data from a table	TRUNCATE TABLE

Creating a Table

- Syntax

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name ( {  
    column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]  
    | table_constraint  
    | LIKE source_table [ like_option [...] ] [, ... ] )  
    [ WITH ( {storage_parameter = value} [, ... ] ) ]  
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]  
    [ COMPRESS | NOCOMPRESS ]  
    [ TABLESPACE tablespace_name ];
```

- Note

- Column-store tables support only **PARTIAL CLUSTER KEY** table-level constraints, but do not support primary and foreign key table-level constraints.
- Only the **NULL**, **NOT NULL**, and **DEFAULT** constant values can be used as column-store table constraints.

Example

- Create a table named **bank_card**.

```
postgres=# CREATE TABLE bank_card1( b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20),b_c_id INT NOT NULL);
```

- Create a temporary table.

```
postgres=# CREATE TEMPORARY TABLE bank_card2(b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20),b_c_id INT NOT NULL);
```

Modifying Table Attributes

- **ALTER TABLE** changes the definition of a table by changing, adding, or deleting columns and constraints. With **ALTER TABLE**, you can:
 - Add, delete, modify, or rename columns.
 - Add or delete constraints.
 - Enable or disable constraints.
 - Change the name of a partition.
 - Modify partition tablespaces.

Modifying Table Attributes

- Syntax

```
ALTER TABLE [ schema_name. ]table_name
{ alter_table_properties
| column_clauses
| references_clause
| constraint_clauses
| partition_clauses
| enable_disable_clause
| set_interval_clause
}
```

- Note

- To run this statement, you must have the **ALTER ANY TABLE** system permission. Common users cannot modify objects of system users.
- When you add a column to a table, ensure that there is no record in the table.
- When you modify a column, ensure that all the values in this column are **NULL**.

Example

- Add the **full_masks** column to the **Bank_card** table.

```
postgres=# ALTER TABLE bank_card ADD full_masks INTEGER;
```

- Change the data type of a column.

```
postgres=# ALTER TABLE bank_card MODIFY full_masks NVARCHAR2(20);
```

- Add a constraint.

```
postgres=# ALTER TABLE bank_card ADD CONSTRAINT ck_bank_card CHECK(b_c_id>0);
postgres=# ALTER TABLE bank_card ADD CONSTRAINT uk_bank_card UNIQUE(full_masks);
```

- Delete the **full_masks** column.

```
postgres=# ALTER TABLE bank_card DROP full_masks;
```

Deleting a Table

- Syntax

```
DROP TABLE [ IF EXISTS ] { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ];
```

- Example

```
postgres=# DROP TABLE IF EXISTS training;
```

- Note

- **DROP TABLE** forcibly deletes a specified table and the indexes depending on the table. After the table is deleted, no functions or stored procedures that need to use this table can be executed.
- Deleting a partitioned table also deletes all partitions in the table.
- Only the table owner or a user with the **DROP** permission can run **DROP TABLE**. The system administrator has this permission by default.

Defining a Partitioned Table

- openGauss supports range partitioning, list partitioning, and hash partitioning.
- A partitioned table has the following advantages over an ordinary table:
 - High query performance: Users can specify partitions when querying a partitioned table, improving query efficiency.
 - High availability: If a partition in a partitioned table is faulty, data in the other partitions is still available.
 - Easy maintenance: If a partition in a partitioned table is faulty, only this partition needs to be repaired.
 - Balanced I/O: Partitions can be mapped to different disks to balance I/O and improve the overall system performance.

Function	SQL Statement
Creating a partitioned table	CREATE TABLE PARTITION
Modifying partitioned table attributes	ALTER TABLE PARTITION
Deleting a partitioned table	DROP TABLE

Creating a Partitioned Table

- Syntax

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
  { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
  | table_constraint
  | LIKE source_table [ like_option [...] ][, ... ]
])
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ COMPRESS | NOCOMPRESS ]
[ TABLESPACE tablespace_name ]
PARTITION BY {
  {RANGE (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_name [, ... ] ) ] ]
  ( partition_less_than_item [, ... ] )} |
  {RANGE (partition_key) [ INTERVAL ('interval_expr') [ STORE IN (tablespace_name [, ... ] ) ] ] ( partition_start_end_item [, ... ] )
  |
  {LIST | HASH (partition_key) (PARTITION partition_name [VALUES (list_values_clause)] opt_table_space )}
} [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

Example

- Create a tablespace.

```
postgres=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
postgres=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
postgres=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
postgres=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
```

- Create a partitioned table.

```
postgres=# CREATE TABLE training(staff_id INTEGER NOT NULL, course_name CHAR(20), course_period DATETIME,
exam_date DATETIME, score INTEGER)
TABLESPACE example1
PARTITION BY RANGE(staff_id)
(
    PARTITION training1 VALUES LESS THAN(100),
    PARTITION training2 VALUES LESS THAN(200),
    PARTITION training3 VALUES LESS THAN(300),
    PARTITION training4 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

Modifying a Partitioned Table

- Syntax

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )} action  
[, ...];
```

- Example

- Modify the tablespace of the **training1** partition.

```
postgres=# ALTER TABLE training MOVE PARTITION training1 TABLESPACE example3;
```

- Rename the **training1** partition to **training5**.

```
postgres=# ALTER TABLE training RENAME PARTITION training1 TO training5;
```

- Merge the **training1** and **training2** partitions into **training3**.

```
postgres=# ALTER TABLE training MERGE PARTITION training1,training2 INTO PARTITION training3;
```

- Delete the **training4** partition.

```
postgres=# ALTER TABLE training DROP PARTITION training4;
```

Deleting a Partitioned Table

- Syntax

```
DROP TABLE [ IF EXISTS ] { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ];
```

- Example

- Delete a partitioned table.

```
postgres=# DROP TABLE training;
```

- Delete a tablespace.

```
postgres=# DROP TABLESPACE example1;
postgres=# DROP TABLESPACE example2;
postgres=# DROP TABLESPACE example3;
postgres=# DROP TABLESPACE example4;
```

Defining an Index

- An index is a sequence of values in one or more columns in a database table. It is a data structure that improves the speed of data access to specific information in a database table. The following table lists the involved SQL statements.

Function	SQL Statement
Creating an index	CREATE INDEX
Modifying index attributes	ALTER INDEX
Deleting an index	DROP INDEX

- Category:
 - Indexes are classified into single-column indexes and multi-column indexes.
 - Indexes are also classified into common indexes, unique indexes, function indexes, and partitioned indexes, depending on how they are used.

Creating an Index

- Description
 - **CREATE INDEX** creates an index in a specified table. Indexes are primarily used to improve database query performance. However, be aware that inappropriate use may compromise performance.
- Note
 - To run this statement, you must have the **CREATE INDEX** or **CREATE ANY INDEX** system permission. Common users cannot create system users' objects.
- Syntax

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.]index_name ] ON table_name [ USING method ]
({ { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] }, ... )
[ WITH ( {storage_parameter = value} [, ...] ) ]
[ TABLESPACE tablespace_name ]
[ WHERE predicate ];
```

Creating an Index

- Create an index in the **bank_card** table.

```
-- Create the ordinary table bank_card.  
postgres=# CREATE TABLE bank_card( b_number NCHAR(30) PRIMARY KEY, b_type NCHAR(20),b_c_id INT NOT NULL);  
-- Create the idx_bank_card index.  
postgres=# CREATE INDEX idx_bank_card ON bank_card(b_number ASC, b_type);
```

- Create a partitioned index in the partitioned table **education**.

```
-- Create the education partitioned table.  
postgres=# CREATE TABLE education(staff_id INTEGER NOT NULL, higest_degree CHAR(8), graduate_school  
VARCHAR(64),graduate_date DATETIME, education_note VARCHAR(70))  
PARTITION BY LIST(higest_degree)  
(  
PARTITION doctor VALUES ('Doctor'),  
PARTITION master VALUES ('Master'),  
PARTITION undergraduate VALUES ('Bachelor')  
);  
-- Create a partitioned index with the name of the index partition specified.  
postgres=# CREATE INDEX idx_education ON education(staff_id ASC, higest_degree) LOCAL (PARTITION doctor, PARTITION  
master, PARTITION undergraduate);
```

Modifying Index Attributes

- Syntax:

```
ALTER INDEX [ schema_name. ]index_name
{ rebuild_clauses
| rename_clauses
| modify_clauses
}
```

- Notes:

- rebuild_clauses
 - **REBUILD [PARTITION index_partition_name]**: Rebuilds a table index or an index partition.
- rename_clauses
 - **RENAME TO [schema_name.] index_name_new**: Specifies the name of the index to be renamed.
- modify_clauses
 - **SET TABLESPACE tablespace_name**: Modifies the tablespace to which a table index belongs.

Example

- Example:

- Rebuild an index.

```
postgres=# ALTER INDEX idx_bank_card REBUILD;
```

- Rename the existing index.

```
postgres=# ALTER INDEX idx_bank_card RENAME TO idx_bank_card_temp;
```

- Set the index to be unavailable.

```
postgres=# ALTER INDEX idx_bank_card UNUSABLE;
```

Deleting an Index

- Syntax

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] index_name [, ...] [ CASCADE | RESTRICT ];
```

- Parameter Description

- IF EXISTS

- Does not report an error if an index does not exist.

- index_name

- Specifies the name of the index to be deleted.

- CASCADE | RESTRICTON

- **CASCADE**: Automatically deletes all objects that depend on the index to be deleted.

- **RESTRICT** (default): Refuses to delete the index if any objects depend on it.

- Example

```
postgres=# DROP INDEX idx_bank_card;
```

Defining a View

- A view is a virtual table exported from one or more base tables, and is used to control user access to data. The following table lists the related SQL statements.

Function	SQL Statement
Creating a view	CREATE VIEW
Deleting a view	DROP VIEW

- Notes:
 - A view is different from the base table. A database only stores the definition of a view and not its data, which is still stored in the original base table.
 - If data in the base table changes, the data queried from the view changes accordingly.
 - In this sense, a view is like a window through which users can see relevant data and changes within the database.

Creating a View

- Syntax

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
[ WITH ( {view_option_name [= view_option_value]} [, ...] ) ] AS query;
```

- Example:

- Create the **privilege_view** view. If the view exists, update it.

```
postgres=# CREATE OR REPLACE VIEW privilege_view AS SELECT b_number, b_type FROM bank_card;
```

- Query the data in the view. The syntax is the same as that for querying a table.

```
postgres=# SELECT * FROM privilege_view;
```

- Query the view structure.

```
postgres=# \d privilege_view;
```

Deleting a View

- Syntax

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```

- Parameter Description

- IF EXISTS

- Deletes a view if it exists.

- view_name

- Specifies the view to be deleted.

- CASCADE | RESTRICT

- **CASCADE**: Automatically deletes all objects that depend on the view (such as other views).

- **RESTRICT**: Refuses to delete the view if any objects depend on it. This is the default value.

- Example

```
postgres=# DROP VIEW IF EXISTS privilege_view;
```

Defining a Sequence

- A sequence can generate numbers with the same interval, and these are used as primary key values. When a sequence value is generated, the sequence is incremented. The following table lists the SQL statements involved.

Function	SQL Statement
Creating a sequence	CREATE SEQUENCE
Modifying sequence attributes	ALTER SEQUENCE
Deleting a sequence	DROP SEQUENCE

Creating a Sequence

- Description
 - Adds a sequencer to the current database. The current user is the owner of the sequencer.
- Syntax

```
CREATE SEQUENCE name
[ INCREMENT [ BY ] increment ]
[ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
[ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
[ OWNED BY { table_name.column_name | NONE }];
```

- Precautions
 - A sequence is a special table that stores arithmetic progressions. It has no actual meaning and is usually used to generate unique identifiers for rows or tables.
 - If a schema name is given, the sequence is created in the specified schema; otherwise, it is created in the current schema. The sequence name must be different from the names of other sequences, tables, indexes, views in the same schema.
 - After the sequence is created, functions **nextval()** and **generate_series(1,N)** insert data to the table. Make sure that the number of times **nextval** is invoked is greater than or equal to N + 1. Otherwise, errors will be reported because the number of times the **generate_series()** function is invoked is N + 1.

Example

- >Create the **seq_auto_extend** sequence starting with **10**, and with **increment** set to **2**, **MAXVALUE** set to **200**, and **CYCLE** specified.

```
postgres=# CREATE SEQUENCE seq_auto_extend START WITH 10 MAXVALUE 200 INCREMENT BY 2 CYCLE;
```

- Obtain the next value of the sequence.

```
postgres=# SELECT nextval ('seq_auto_extend');
```

- Use the sequence for ID auto-increment.

```
postgres=# CREATE SEQUENCE serial1
START 101
CACHE 20
;
```

```
CREATE TABLE test (id number(6) default nextval ('serial1'), name varchar(20),constraint ts_id primary key(id));
```

Modifying Sequence Attributes

- Syntax

- Modify the owning column of a sequence.

```
ALTER SEQUENCE [ IF EXISTS ] name  
[MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]  
[ OWNED BY { table_name.column_name | NONE } ];
```

- Change the owner of a sequence.

```
ALTER SEQUENCE [ IF EXISTS ] name OWNED TO new_owner;
```

- Example

- Modify the **seq_auto_extend** sequence.

```
-- Set INCREMENT BY to 4 and MAXVALUE to 400.  
postgres=# ALTER SEQUENCE seq_auto_extend MAXVALUE 400 INCREMENT BY 4 CYCLE;
```

- The column to which the **seq_auto_extend** sequence belongs is changed to **b_number**.

```
postgres=# ALTER SEQUENCE seq_auto_extend OWNED BY bank_card.b_number;
```

Deleting a Sequence

- Syntax

```
DROP SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```

- Parameter Description

- IF EXISTS

Does not report an error if the sequence to be deleted does not exist.

- I [schema_name.] sequence_name

Specifies the sequence to be deleted.

- **CASCADE** automatically deletes objects that depend on the sequence to be deleted.

- **RESTRICT** refuses to delete the sequence if any objects depend on it. This is the default action.

- Example

- Delete the **seq_auto_extend** sequence.

```
postgres=# DROP SEQUENCE IF EXISTS seq_auto_extend;
```

Quiz

1. (True or false) When an index is created for a table, the corresponding base table is deleted if the index is canceled.
 - A. True
 - B. False
2. (Single-choice) SQL integrates data query, data manipulation, data definition, and data control. Which of the following functions is implemented by the CREATE, DROP, and ALTER statements?
 - A. Data query
 - B. Data manipulation
 - C. Data definition
 - D. Data control

Quiz

3. (Multiple-choice) Create a decremental sequence named **seq_1**. The starting value is 400, the step is -4 , and the minimum value is 100. When the sequence reaches the minimum value, it can be cyclically executed. Which of the following statements are correct?
- A. CREATE SEQUENCE seq_1 START WITH 400 MAXVALUE 100 INCREMENT BY -4 CYCLE;
 - B. CREATE SEQUENCE seq_1 MAXVALUE 400 MINVALUE 100 INCREMENT BY -4 CYCLE;
 - C. CREATE SEQUENCE seq_1 START WITH 400 MINVALUE 100 INCREMENT BY -4 CYCLE;
 - D. CREATE SEQUENCE seq_1 START WITH 400 MINVALUE 100 MAXVALUE 400 INCREMENT BY -4 CYCLE;

Contents

1. SQL Syntax Quick Start
2. SQL Syntax Classification
- 3. Operators and Common Functions**

Logical Operators

- Common logical operators include AND, OR, and NOT. The operation result can be **TRUE**, **FALSE**, or **NULL** (which means unknown). Their priorities are NOT > AND > OR.

Operator	Function
and	The logical AND operator can be used in query conditions, such as WHERE , ON , and HAVING .
or	The logical OR operator can be used in query conditions, such as WHERE , ON , and HAVING .
not	The NOT keyword can be added before the condition expression after the WHERE or HAVING clause, in order to reverse the condition result. This keyword is used together with the relational operation, for example, not in and not exists .

```
-- Search the bank_card table for the information about a credit card whose customer ID is 1.  
postgres=# SELECT * FROM bank_card where b_type='Credit Card' and b_c_id=1;  
-- Search the bank_card table for the information about a credit card whose customer ID is 1.  
postgres=# SELECT * FROM bank_card where b_type='Credit Card' or b_c_id=1;
```

Comparison Operators

- Comparison operators are available for the most data types and return Boolean values.

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<> or !=	Not equal to

```
-- Search the bank_card table for information about a credit card.  
postgres=# SELECT * FROM bank_card where b_type='Credit Card' ;  
-- Search the bank_card table for information about a non-credit card.  
postgres=# SELECT * FROM bank_card where b_type<>'Credit Card';
```

Character Processing Functions and Operators

- **instr(string1,string2,int1,int2)** returns the text from **int1** to **int2** in **string1**. The first **int** indicates the start position for matching, and the second **int** indicates the number of matching times.

```
postgres=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
```

```
Instr  
-----  
6
```

- **overlay(string placing string FROM int [for int])** replaces a substring. **FROM int** indicates the start position of the replacement in the first string. **for int** indicates the number of characters replaced in the first string.

```
postgres=# SELECT overlay('hello' placing 'world' FROM 2 for 3 );
```

```
overlay  
-----  
hworldo
```

Character Processing Functions and Operators

- **substring(string [FROM int] [for int])** extracts a substring. **FROM int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

```
postgres=# SELECT substring('Thomas' FROM 2 for 3);
Substring
-----
hom
(1 row)
```

- **replace(string text, FROM text, to text)** replaces all occurrences in **string** of substring **FROM** with substring **to**.

```
postgres=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
replace
-----
abXXXefabXXXef
(1 row)
```

Common Character Processing Functions

Character Processing Functions	Function	Example
basicemailmasking(text, char DEFAULT 'x'::bpchar)	Masks text before the first at sign (@).	basicemailmasking('abcd@gmail.com') xxxx@gmail.com
char_length(string)	Specifies the number of characters in a string.	char_length('hello') 5
lengthb(text/bpchar)	Obtains the number of bytes in a specified string.	lengthb('hello') 5
left(str text, n int)	Returns the first <i>n</i> characters in a string. When <i>n</i> is negative, all but the last n characters are returned.	left('abcde', 2) ab
right(str text, n int)	Returns the last <i>n</i> characters in a string. When <i>n</i> is negative, all but the first n characters are returned.	right('abcde', 2) de
lpad(string text, length int [, fill text])	Fills up string to length by appending the character fill (a space by default). If string is already longer than length , it is truncated.	lpad('hi', 5, 'xyza') xyzhi
rpad(string text, length int [, fill text])	Fills up string to length by appending the character fill (a space by default). If string is already longer than length , it is truncated.	rpad('hi', 5, 'xy') hixyx
notlike(x bytea name text, y bytea text)	Compares x and y to check if they are inconsistent.	notlike(1,2) t notlike(1,1) f
rawcat(raw,raw)	Indicates the string concatenation functions.	rawcat('ab','cd') ABCD
reverse(str)	Returns reversed strings.	reverse('abcde') edcba

Arithmetic Functions and Operators

Arithmetic Operator	Description	Arithmetic Operator	Description
+	Addition	!!	Factorial (prefix operator)
-	Subtraction		Binary OR
*	Multiplication	&	Binary AND
/	Division (The result is not rounded.)	#	Binary XOR
%	Modulo	~	Binary NOT
@	Absolute value	^	Power (exponent calculation)
	Square root	<<	Left shift
	Cubic root	>>	Right shift

```
postgres=# SELECT 2+3,2*3, @ -5.0, 2.0^3.0, | / 25.0, 91&15, 17#5,1<<4 AS RESULT;
```

2+3		2*3		@ -5.0		2.0^3.0		/ 25.0		91&15		17#5		1<<4
5		6		5.0		8.00000		5		11		20		16

Example

- **abs(exp), cos(exp), sin(exp), acos(exp), and asin(exp)** return the absolute value, cosine value, sine value, arc cosine value, and arc sine value of an expression, respectively.
 - The return value type of **abs(exp)** is the same as that of **exp**. The return values of the other parameters are numbers.
 - The input parameter **exp** of **asin** and **acos** is an expression that can be converted into a numeric value. The value range is $[-1,1]$.

```
postgres=# SELECT abs(-10),cos(0),sin(0),acos(1),asin(0);
```

ABS	COS	SIN	ACOS	ASIN
10	1	0	0	0

```
1 rows fetched.
```

Example

- **bitand(exp1,exp2)** performs an AND (&) operation on two digits.

```
postgres=# SELECT bitand(29,15);
```

```
BITAND(29,15)
```

```
-----  
13
```

- **round(number[, decimals])** truncates a number value to the left or right of a specified decimal point.

```
postgres=# SELECT round(1234.5678,-2),round(1234.5678,2);
```

```
ROUND(1234.5678,-2) | ROUND(1234.5678,2)
```

```
-----+-----  
1200      1234.57
```

Common Arithmetic Functions

Arithmetic Functions	Function	Example
ceil(x)	Not smaller than the minimum integer of a parameter	ceil(-42.8) -42
floor(x)	Not larger than the maximum integer of a parameter	floor(-42.8) -43
log(x)	Logarithm with 10 as the base	log(100.0) 2.000000000000000
div(y numeric, x numeric)	Integer part of y/x	div(9,4) 2
trunc(x)	Truncates and retains the integral part.	trunc(42.8) 42
cbrt(dp)	Cubic root	cbrt(27.0) 3
mod(x,y)	Remainder of x/y (modulus) If x equals to 0, 0 is returned.	mod(9,4) 1
power(a double precision, b double precision)	b power of a	power(9.0, 3.0) 729.0000000000000

Time and Date Operators (+)

- Time and date operators (+)

```
postgres=# SELECT date '2021-5-28' + integer '7' AS RESULT;  
          result  
-----
```

```
2021-06-04 00:00:00
```

```
postgres=# SELECT date '2021-05-28' + interval '1 hour' AS RESULT;  
          result  
-----
```

```
2021-05-28 01:00:00
```

```
postgres=# SELECT date '2021-05-28' + time '03:00' AS RESULT;  
          result  
-----
```

```
2021-05-28 03:00:00
```

```
postgres=# SELECT interval '1 day' + interval '1 hour' AS RESULT;  
          result  
-----
```

```
1 day 01:00:00
```

Time and Date Operators (-)

- Time and date operators (-)

```
postgres=# SELECT date '2021-05-01' - date '2021-04-28' AS RESULT;  
          result  
-----  
            3 days
```

```
postgres=# SELECT date '2021-05-01' - integer '7' AS RESULT;  
          result  
-----  
2021-04-24 00:00:00
```

```
postgres=# SELECT date '2021-05-28' - interval '1 hour' AS RESULT;  
          result  
-----  
2021-05-27 23:00:00
```

```
postgres=# SELECT time '05:00' - interval '2 hours' AS RESULT;  
          result  
-----  
03:00:00
```

Time and Date Operators (* and /)

- Time and date operators (*)

```
postgres=# SELECT 900 * interval '1 second' AS RESULT;  
result  
-----  
00:15:00
```

```
postgres=# SELECT 21 * interval '1 day' AS RESULT;  
result  
-----  
21 days
```

```
postgres=# SELECT double precision '3.5' * interval '1 hour' AS RESULT;  
result  
-----  
03:30:00
```

- Time and date operators (/)

```
postgres=# SELECT interval '1 hour' / double precision '1.5' AS RESULT;  
result  
-----  
00:40:00
```

Time and Date Processing Functions

Time/Date Function	Function	Example
age(timestamp, timestamp)	Subtracts parameters, producing a result in YYYY-MM-DD format. If the result is negative, the returned result is also negative. The input parameters can also contain the timezone if required.	age(timestamp '2001-04-10', timestamp '1957-06-13') 43 years 9 mons 27 days
age(timestamp)	Subtracts the current time with the parameter. The input parameter can also contain the timezone if required.	age(timestamp '1957-06-13') 64 years 2 mons 18 days
current_date	Specifies the current date.	current_date 2021-05-01
date_part(text, timestamp)	Obtains the value of a subdomain in date or time (for example, the year or hour).	date_part('hour', timestamp '2001-02-16 20:38:40') 20
trunc(timestamp)	Truncates to day by default.	trunc(timestamp '2001-02-16 20:38:40') 2001-02-16 00:00:00
sysdate	Specifies the current date and time.	sysdate 2021-05-01 17:04:49
justify_days(interval)	Adjusts intervals to 30-day time periods, which are represented as months.	justify_days(interval '35 days') 1 mon 5 days
pg_sleep(seconds)	Specifies the delay time of the server thread, in seconds.	pg_sleep(10)
last_day(d)	Returns the date of the last day of the month that contains <i>date</i> .	last_day(to_date('2017-01-01', 'YYYY-MM-DD')) 2017-01-31 00:00:00

Type Conversion Functions

- **to_char(int, text), to_clob(str), to_date(exp[, fmt]), and to_number(n[, fmt])** convert the specified input parameter to the CHAR, CLOB, DATE, and NUMBER types, respectively.

```
postgres=# SELECT to_char(125, '999'),to_clob('hello111'::CHAR(15)),to_date('05 Dec 2000', 'DD Mon YYYY'),  
to_number('12,454.8-', '99G999D9S');
```

TO_CHAR	TO_CLOB	TO_DATE	TO_NUMBER
125	hello111	2000-12-05 00:00:00	-12454.8

Type Conversion Functions

Type Conversion Function	Function	Example
cast(x as y)	Converts x into the type specified by y.	cast('22-oct-1997' as timestamp) 1997-10-22 00:00:00
hextoraw(string)	Converts a string in hexadecimal format into binary format.	hextoraw('7D') 7D
numtoday(numeric)	Converts values of the NUMBER type into the timestamp of a specified type.	numtoday(2) 2 days
pg_systimestamp()	Obtains the system timestamp.	pg_systimestamp() 2021-05-14 11:21:28.317367+08
rawtohex(string)	Converts a string in binary format into hexadecimal format.	rawtohex('1234567') 31323334353637
to_bigint(varchar)	Converts the character type to the BIGINT type.	to_bigint('123364545554455') 123364545554455
to_timestamp(text, text)	Converts values of the string type into the timestamp of a specified type.	to_timestamp('05 Dec 2000', 'DD Mon YYYY') 2000-12-05 00:00:00
convert_to_nocase(text, text)	Converts a string into a specified encoding type.	convert_to_nocase('12345', 'GBK') \x3132333435

Geometric Operators

Geometric Operator	Description	Example
+	Translation	box '((0,0),(1,1))' + point '(2.0,0)' (3,1),(2,0)
-	Translation	box '((0,0),(1,1))' - point '(2.0,0)' (-1,1),(-2,0)
*	Scaling out/Rotation	box '((0,0),(1,1))' * point '(2.0,0)' (2,2),(0,0)
/	Scaling in/Rotation	box '((0,0),(2,2))' / point '(2.0,0)' (1,1),(0,0)
#	Intersection	box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' (1,1),(-1,-1)
@-@	Length or circumference	@-@ path '((0,0),(1,0))' 2
@@	Center	@@ circle '((0,0),10)' (0,0)
<->	Distance	circle '((0,0),1)' <-> circle '((5,0),1)' 3
<^	Is below?	box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' t
>^	Is above?	box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))' t
&&	Overlaps? (One point in common makes this true.)	box '((0,0),(1,1))' && box '((0,0),(2,2))' t
<<	Is strictly left of?	circle '((0,0),1)' << circle '((5,0),1)' t

Geometric Functions

Geometric Function	Description	Example
area(object)	Area	area(box '((0,0),(1,1))') 1
center(object)	Center	center(box '((0,0),(1,2))') (0.5,1)
diameter(circle)	Diameter of a circle	diameter(circle '((0,0),2.0)') 4
height(box)	Vertical size of a rectangle	height(box '((0,0),(1,1))') 1
isclosed(path)	Is it a closed path?	isclosed(path '((0,0),(1,1),(2,0))') t
isopen(path)	Is it an open path?	isopen(path '[(0,0),(1,1),(2,0)]') t
length(object)	Length calculation	length(path '((-1,0),(1,0))') 4
npoints(path)	Number of points in a path	npoints(path '[(0,0),(1,1),(2,0)]') 3
npoints(polygon)	Number of points in a polygon	npoints(polygon '((1,1),(0,0))') 2
pclose(path)	Converts a path to closed	pclose(path '[(0,0),(1,1),(2,0)]') ((0,0),(1,1),(2,0))
popen(path)	Converts a path to open	popen(path '((0,0),(1,1),(2,0))') [(0,0),(1,1),(2,0)]
radius(circle)	Radius of a circle	radius(circle '((0,0),2.0)') 2
width(box)	Horizontal size of a box	width(box '((0,0),(1,1))') 1

Geometric Functions

Geometric Type Conversion Functions	Description	Example
box(circle)	Circle to box	box(circle '((0,0),2.0)') (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
box(point, point)	Point to box	box(point '(0,0)', point '(1,1)')--(1,1),(0,0)
box(polygon)	Polygon to box	box(polygon '((0,0),(1,1),(2,0))') --(2,1),(0,0)
circle(box)	Box to circle	circle(box '((0,0),(1,1))') <(0.5,0.5),0.707106781186548>
circle(point, double precision)	Center and radius to circle	circle(point '(0,0)', 2.0) <(0,0),2>
circle(polygon)	Polygon to circle	circle(polygon '((0,0),(1,1),(2,0))') <(1,0.3333333333333333),0.924950591148529>
lseg(box)	Box diagonal to line segment	lseg(box '((-1,0),(1,0))') [(1,0),(-1,0)]
lseg(point, point)	Point to line segment	lseg(point '(-1,0)', point '(1,0)') [(-1,0),(1,0)]
path(polygon)	Polygon to path	path(polygon '((0,0),(1,1),(2,0))') ((0,0),(1,1),(2,0))
point(box)	Box center	point(box '((-1,0),(1,0))') (0,0)
point(circle)	Circle center	point(circle '((0,0),2.0)') (0,0)

Quiz

1. (Single-choice) What is the result returned by the SQL statement **SELECT date '2021-4-28' + integer '7'
AS RESULT;?**
 - A. 2021-05-05
 - B. 2021-05-05 01:00:00
 - C. 2021-05-05 00:00:00
 - D. 00:00:00
2. (Multiple-choice) Which of the following are arithmetic functions?
 - A. length(str)
 - B. sin(n)
 - C. round(v numeric, s int)
 - D. hex(p1)

Quiz

3. (Multiple-choice) Which of the following are logical operators?
 - A. and
 - B. or
 - C. not
 - D. not or
4. (Multiple-choice) Which three separate approaches to pattern matching are provided by a database?
 - A. LIKE
 - B. Same
 - C. SIMILAR TO
 - D. POSIX regular expressions

Summary

- This chapter describes the concepts, classification, and data types associated with SQL statements.
- This chapter describes the DQL, DML, DDL, and DCL in SQL statements, and provides the syntax, application scenarios, and examples of each statement.
- This chapter describes common system functions, operators, and typical examples of databases.

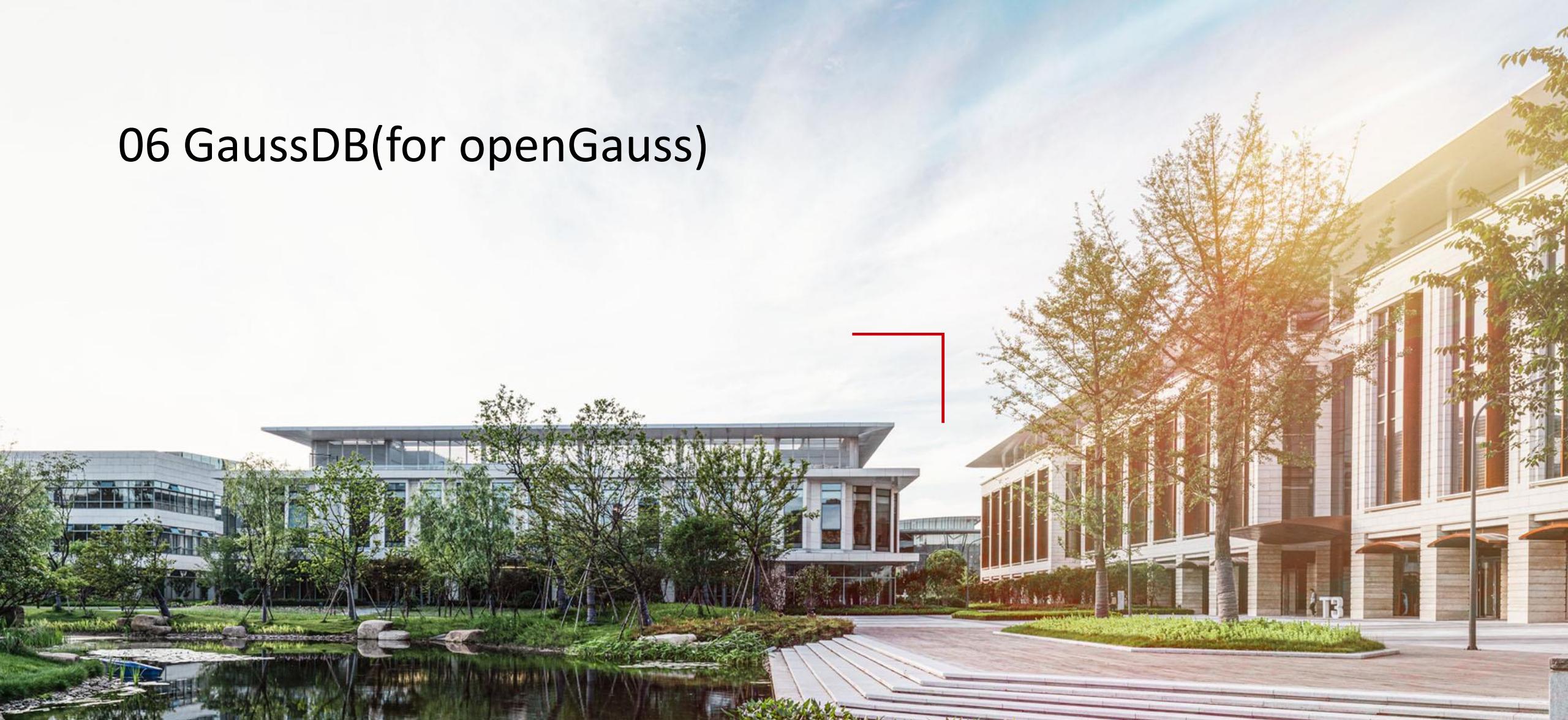
Recommendations

- HUAWEI CLOUD:
 - <https://www.huaweicloud.com/>
- openGauss help document:
 - <https://opengauss.org/en/docs/2.0.0/docs/Quickstart/Quickstart.html>
- openGauss code repository:
 - <https://gitee.com/opengauss>

Acronyms and Abbreviations

Acronym and Abbreviation	Full Spelling
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
DQL	Data Query Language
I/O	Input/output
SQL	Structured Query Language

06 GaussDB(for openGauss)



Foreword

- Databases are essential tools and widely used throughout enterprises. GaussDB(for openGauss) is one of the main application scenarios in the Kunpeng ecosystem. Database technology is the focus of continued innovation, with cloud-based, distributed, and multi-mode processing representing primary future trends. This chapter describes the key advantages, features, and application scenarios of GaussDB(for openGauss), as well as related tools.

Objectives

- Upon completion of this course, you will be familiar with:
 - Basics of GaussDB(for openGauss)
 - Features of GaussDB(for openGauss)
 - Tools and services related to GaussDB(for openGauss)

Contents

- 1. Introduction to GaussDB(for openGauss)**
2. Enterprise-Level Features of GaussDB(for openGauss)
3. Comprehensive Tools and Service-oriented Capabilities
4. Application Scenarios and Cases

Introduction to GaussDB(for openGauss)



GaussDB(for openGauss)

- GaussDB(for openGauss) is an enterprise-level distributed relational database developed by Huawei and based on the openGauss ecosystem. It features Hybrid Transactional/Analytical Processing (HTAP) capabilities and supports intra-city deployment across AZs, scale-out of more than 1,000 nodes, and storage for four petabytes of data to ensure zero data loss. It is highly available, reliable, secure, and scalable, and provides key capabilities including quick deployment, backup, restoration, monitoring, and alarm reporting for enterprises.

In-House, Full-Stack Development

- GaussDB(for openGauss) is an in-house product based on Kunpeng, featuring full-stack, independent, and controllable capabilities. GaussDB(for openGauss) continuously optimizes its performance to meet ever-increasing demands across a wide range of scenarios.
- Fully utilizing Huawei's accumulated R&D capabilities in database kernels, GaussDB(for openGauss) combines the enterprise-level capabilities of traditional relational databases and the advantages of distributed databases on the Internet, and features high availability, scalability, and performance, as well as enhanced data security. In addition, GaussDB(for openGauss) works with Huawei's powerful software and hardware R&D capabilities to achieve full-stack, independent, and controllable capabilities.
- Core technologies:
 - High performance: distributed execution framework, GTM-Lite technology, and NUMA-aware-based transaction processing
 - High availability: cross-AZ/region DR and parallel playback for ultimate Recovery Time Objective (RTO)
 - High scalability: scale-out online

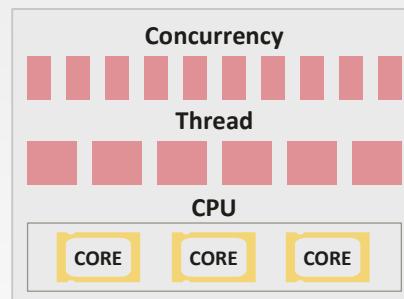
Product Positioning

Centralized Deployment

Focused on core enterprise transactions

GaussDB(for openGauss)

Kunpeng-powered enterprise-level database



Huawei-developed kernel, maximum performance, and enhanced high availability

Distributed Deployment

Focused on massive transaction expansion



Government



Transportation



Finance



Energy



Carrier



1 GB/day

An Internet user



10 GB/day

A smart home



64 TB/day

A self-driving car



200 TB/day

A connected aircraft



1 PB/day

A digitally interconnected factory



50 PB/day

Toutiao

Data infrastructure

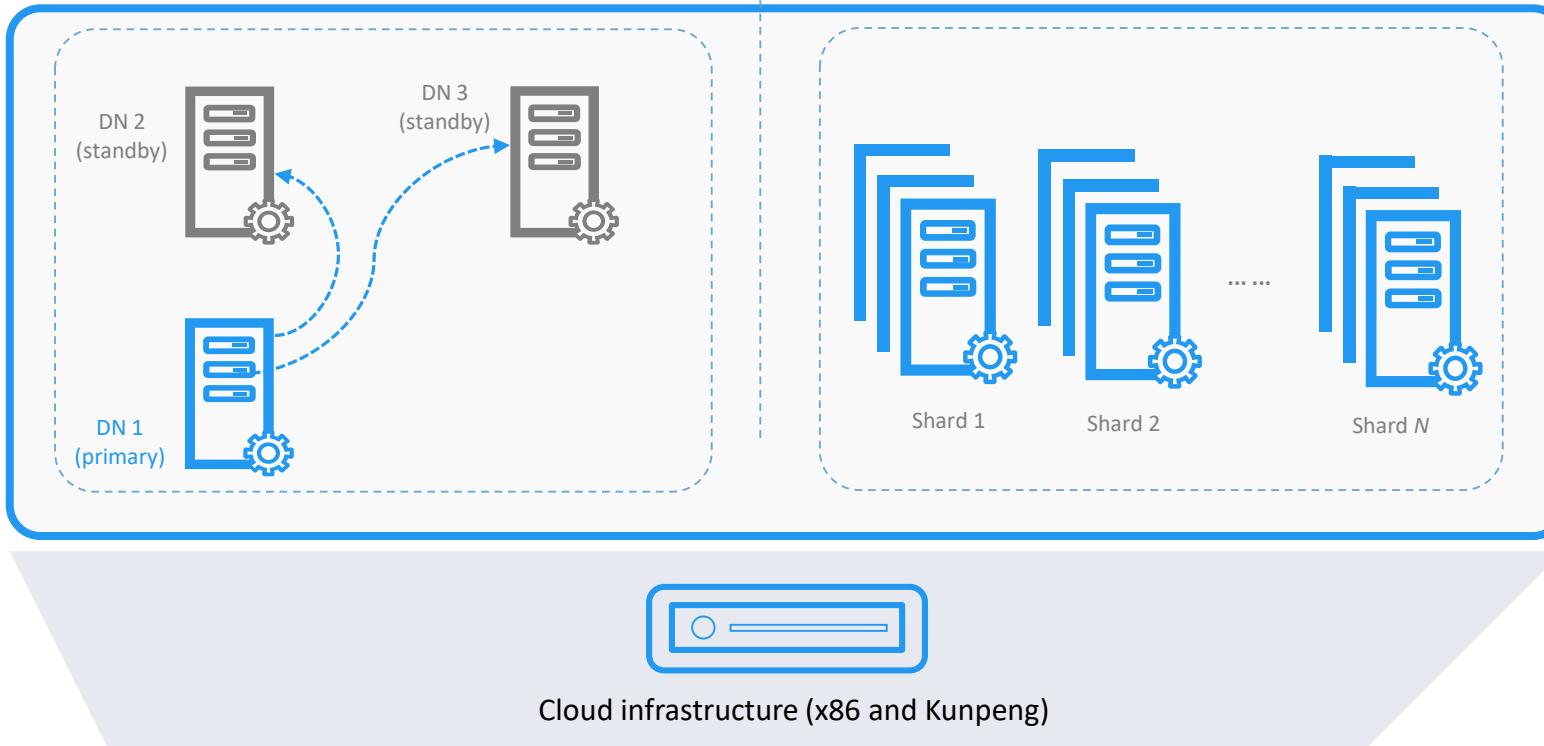
Chip | Storage | Database | Big data | AI

Heterogeneous computing framework: end-to-end multi-architecture computing power

High Availability - Flexible Deployment

Primary/Standby Deployment

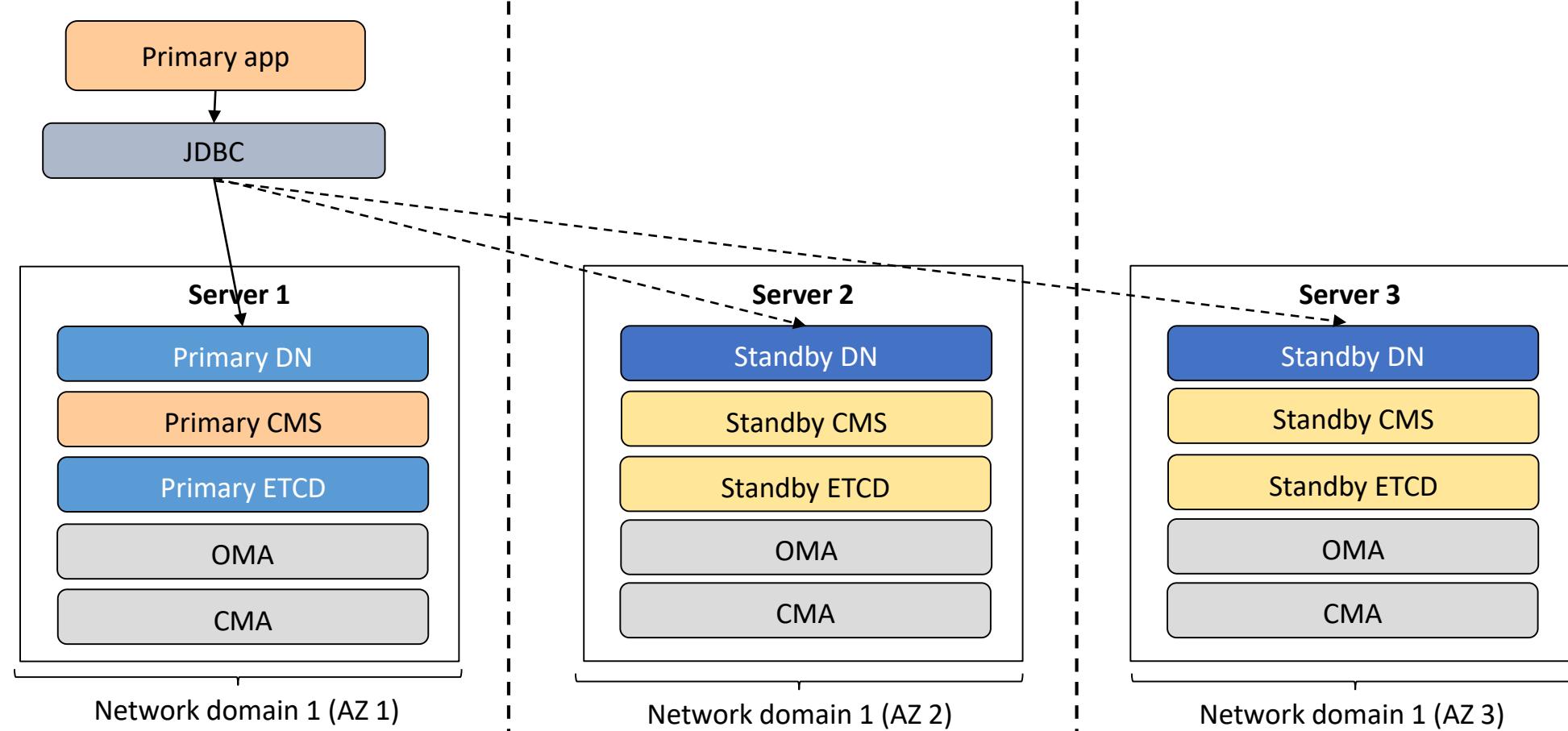
- 1+1 (maximum availability) or 1+2 (maximum protection) primary/standby
- Log replication-based hot backups
- High availability guaranteed



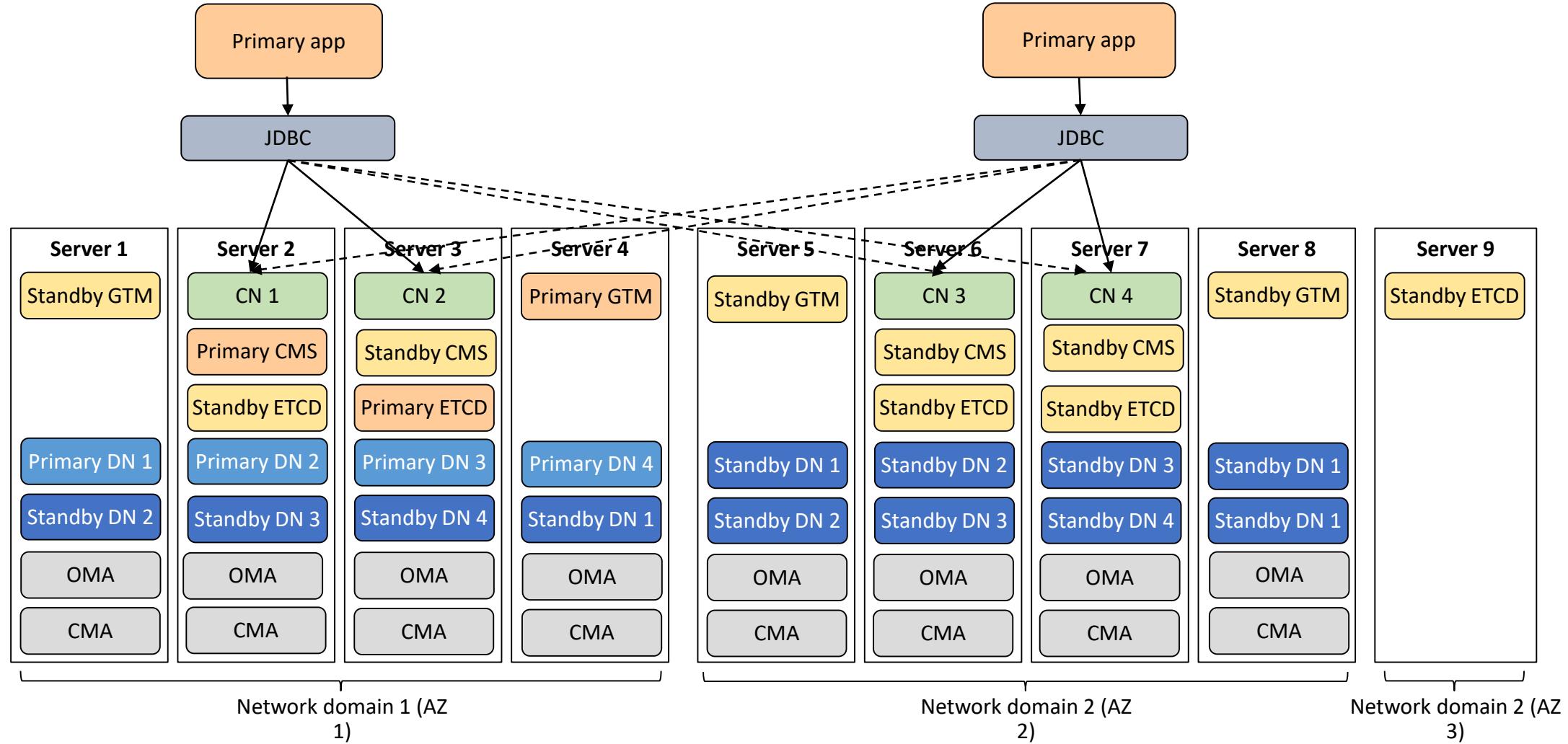
Distributed Deployment

- Distributed high scalability: Read/write requests are offloaded through shards to meet service expansion requirements.
- Distributed high availability: 3DC geo-redundant deployment

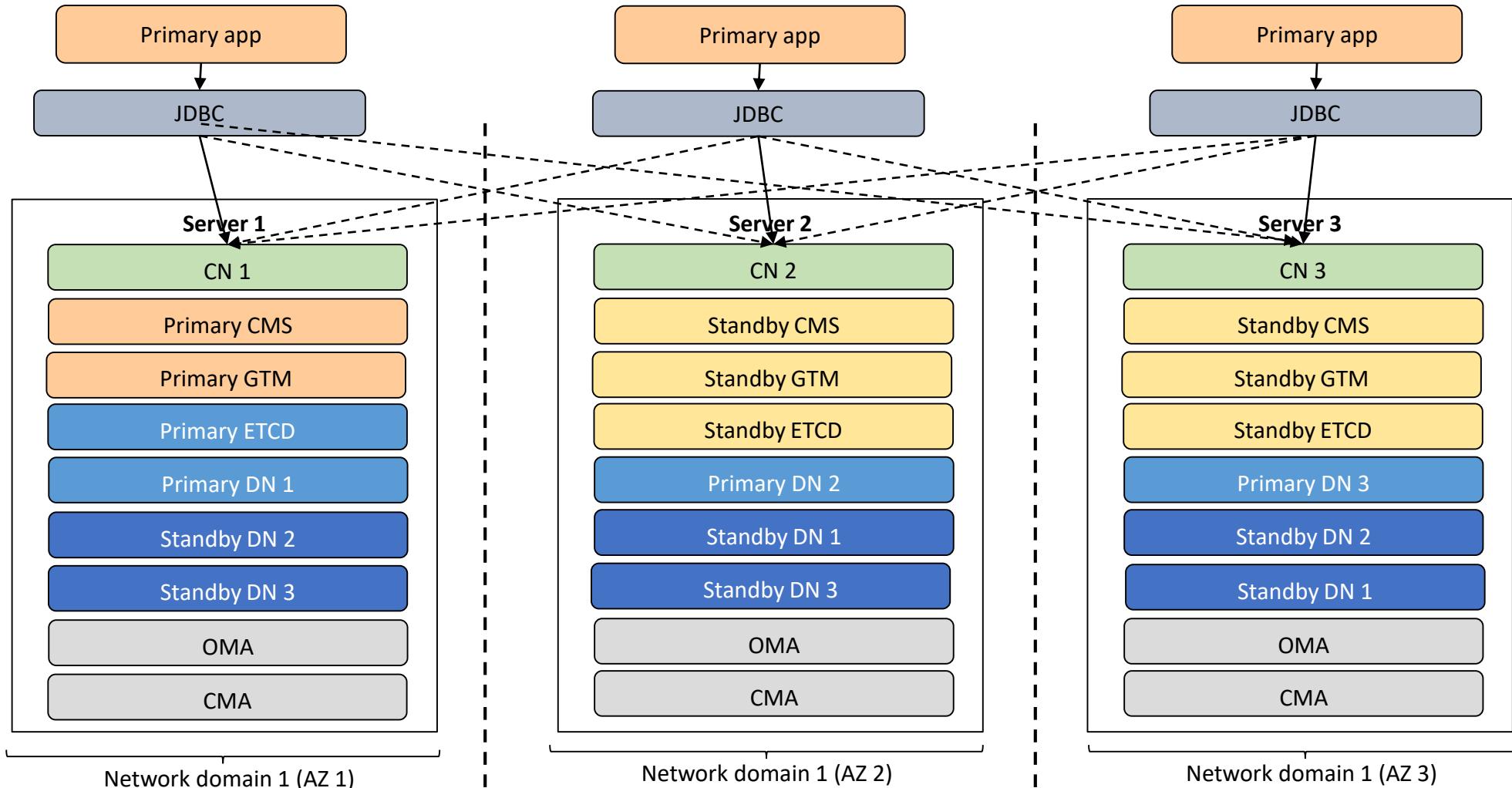
High Availability - Centralized Cluster Deployment



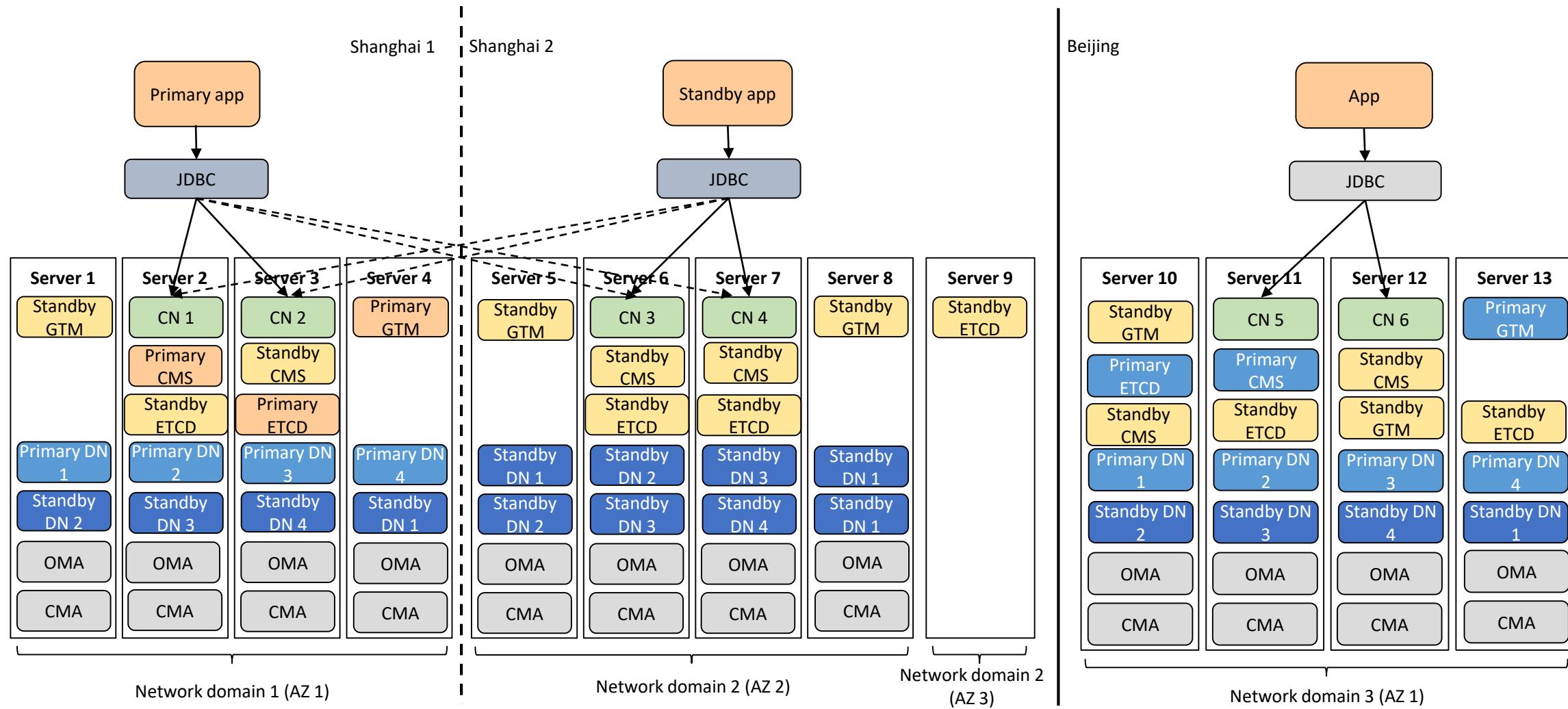
High Availability - Distributed Cluster in Intra-City Active-Active Mode



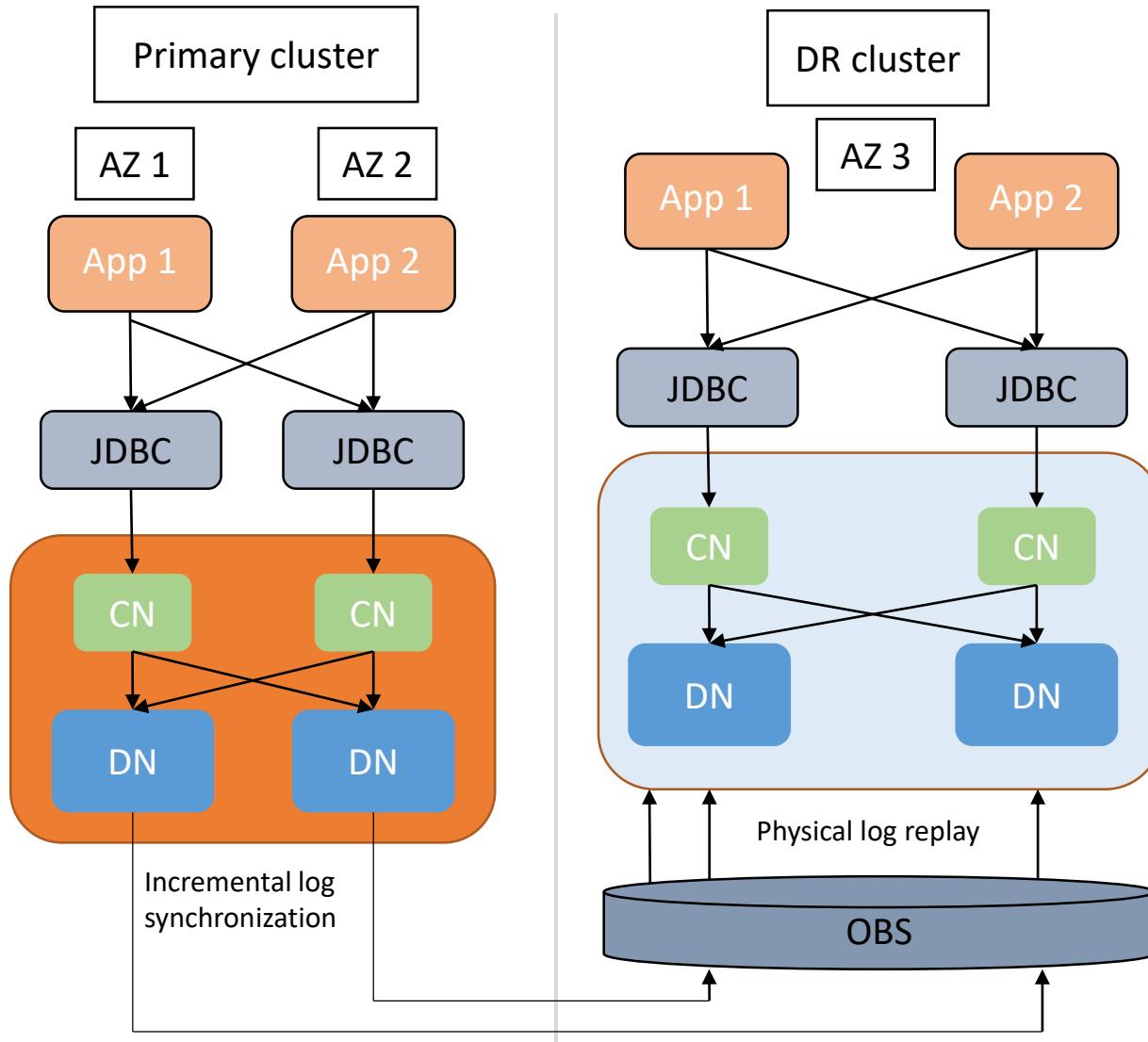
High Availability - Distributed Cluster in Intra-City 3-AZ Multi-active Mode



High Availability - Distributed Cluster in 3DC Geo-redundant Mode



High Availability - Remote DR



Features:

- A and B are deployed in intra-city active-active mode.
- C is a remote DR DC.
- Dual clusters are supported, causing little impact on production.

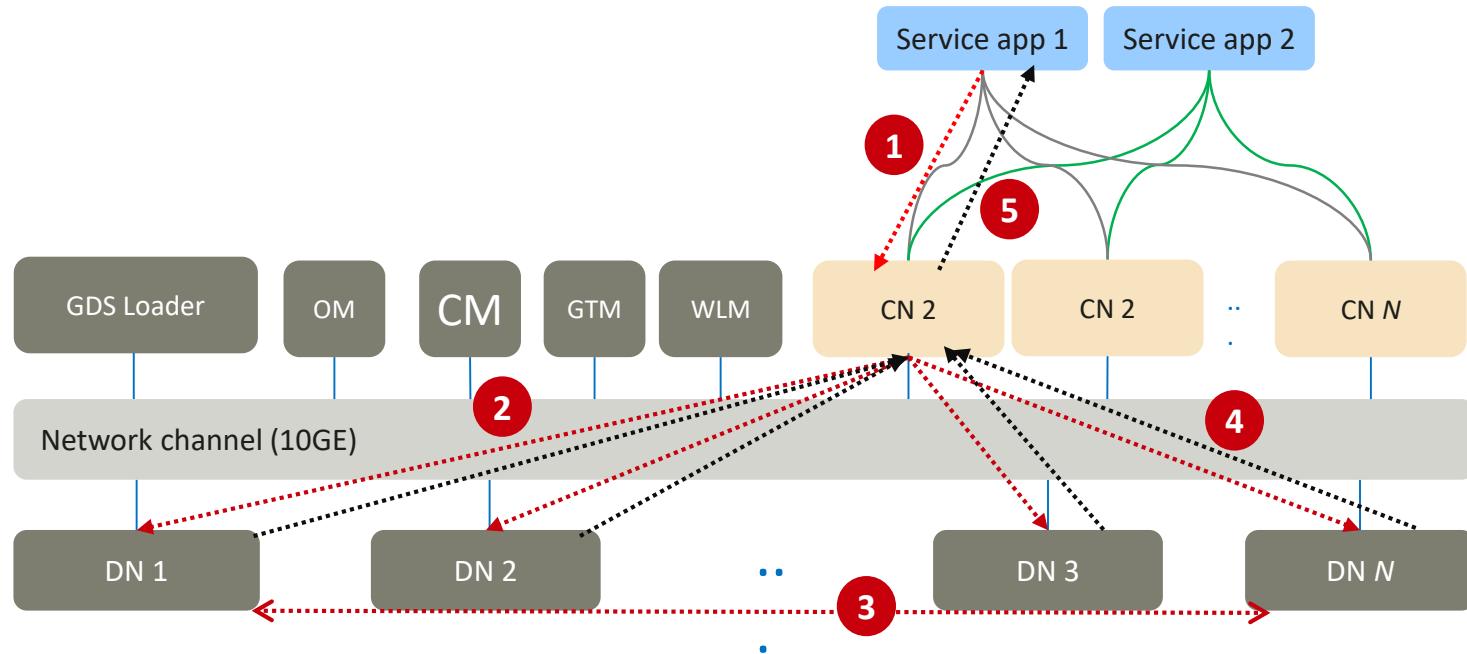
Pain points:

- Resources at the DR backup site are not fully utilized.

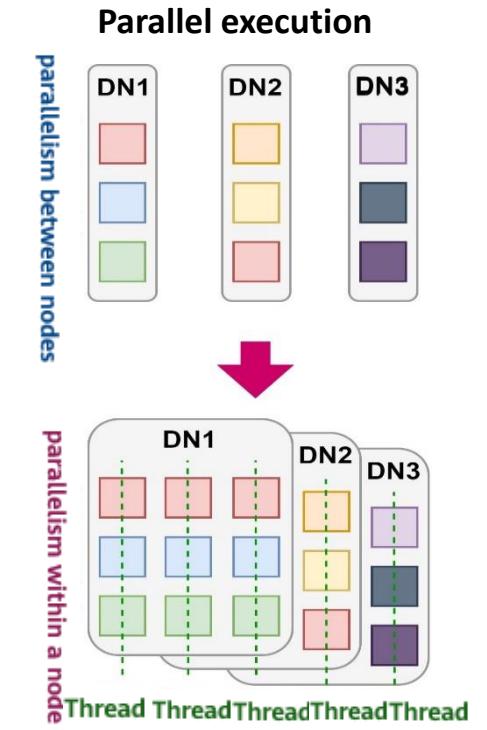
Specifications:

- RPO: 10 seconds
- RTO: 10 minutes

High Performance - Fully Parallel Distributed Execution



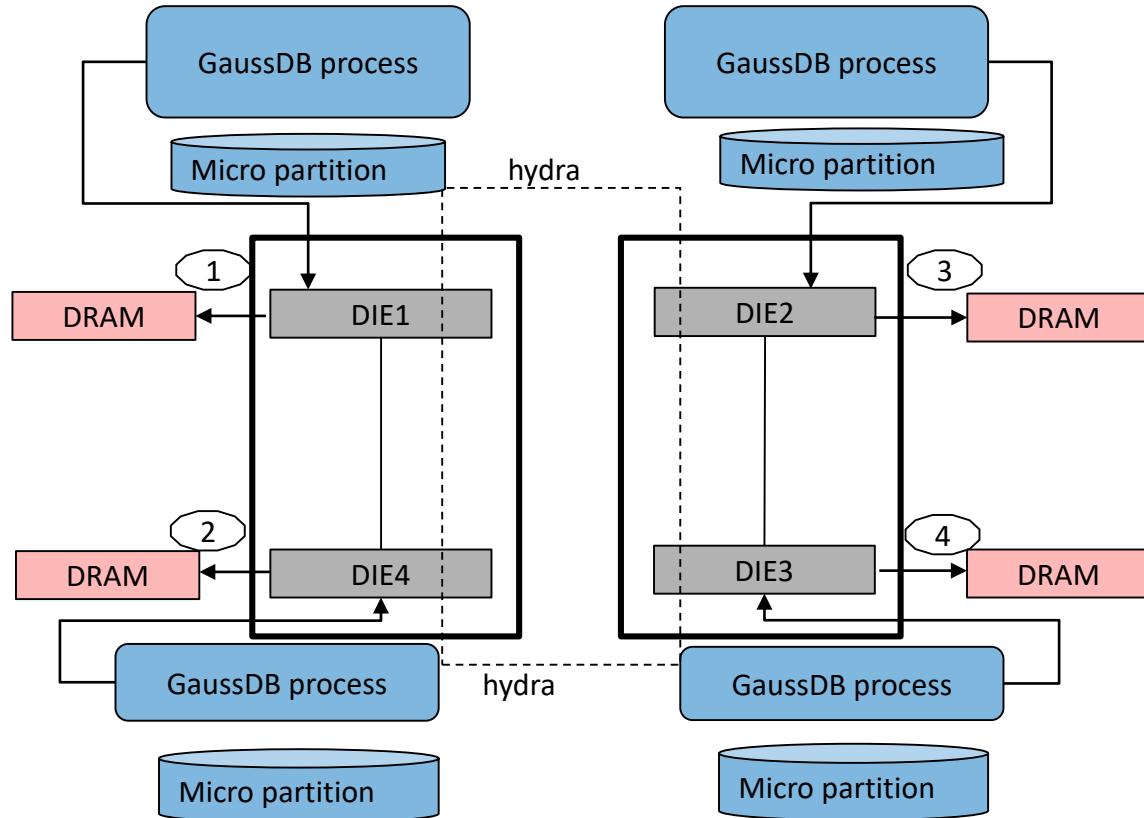
- ① The service application sends an SQL statement to the CN. This SQL statement can be **INSERT, DELETE/DROP, UPDATE, or SELECT**.
- ② The CN uses a database **optimizer** to generate an execution plan. DNs will process data according to this plan.
- ③ Data is evenly distributed on each DN using consistent hashing technology. As such, a DN may need to obtain data from other DNs during data processing. GaussDB provides three types of streams (**Broadcast, Gather, and Redistribution**) to reduce data transfer between DNs.
- ④ The DN returns result sets to the CN.
- ⑤ The CN summarizes and sends the results to the service application.



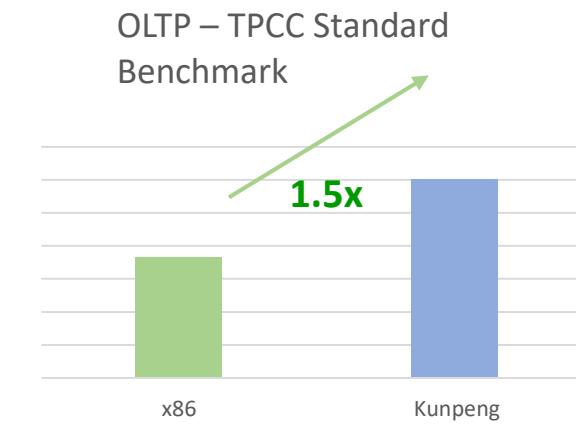
Multi-thread parallel execution

- The current multi-core feature is fully utilized and multi-thread parallel execution is used to improve the system throughput.

High Performance - NUMA-Aware Technology



- Threads are bound to cores to prevent thread offset between cores.
- The NUMA-based data structure is reconstructed to reduce cross-core access.
- Data is partitioned to reduce thread access conflicts.
- The algorithm is adjusted to reduce single-point bottlenecks.
- ARM atomic instructions are used to reduce the computing overhead.



High Performance - Other Features (1)

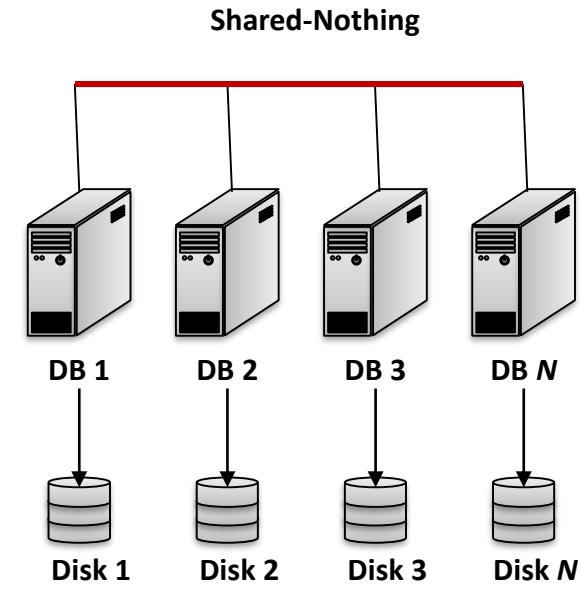
- High-speed parallel data loading
 - You can fully utilize the computing and I/O capabilities of all nodes to achieve the maximum parallel data import speed. Data is imported to GaussDB(for openGauss) using a specific format (CSV, TEXT, or FIXED).
- Symmetric Multi-Processing (SMP) technology
 - The SMP technology of GaussDB(for openGauss) uses the multi-core CPU architecture of a computer to implement multi-thread parallel computing, fully using CPU resources to improve query performance.
 - It significantly reduces the execution time of a single query.
 - It improves system throughput within the same period and effectively improves system resource utilization.

High Performance - Other Features (2)

- Low Level Virtual Machine (LLVM) dynamic compilation technology
 - Based on the query execution plan tree, and with the library functions provided by LLVM, GaussDB(for 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, thereby improving the query performance.
- Adaptive compression
 - Adaptive compression chooses a suitable compression algorithm for data based on the data type and features, achieving high performance in compression ratio, import, and query.
 - GaussDB(for openGauss) has implemented various compression algorithms, including RLE, DELTA, BYTEPACK/BITPACK, LZ4, ZLIB, and LOCAL DICTIONARY.

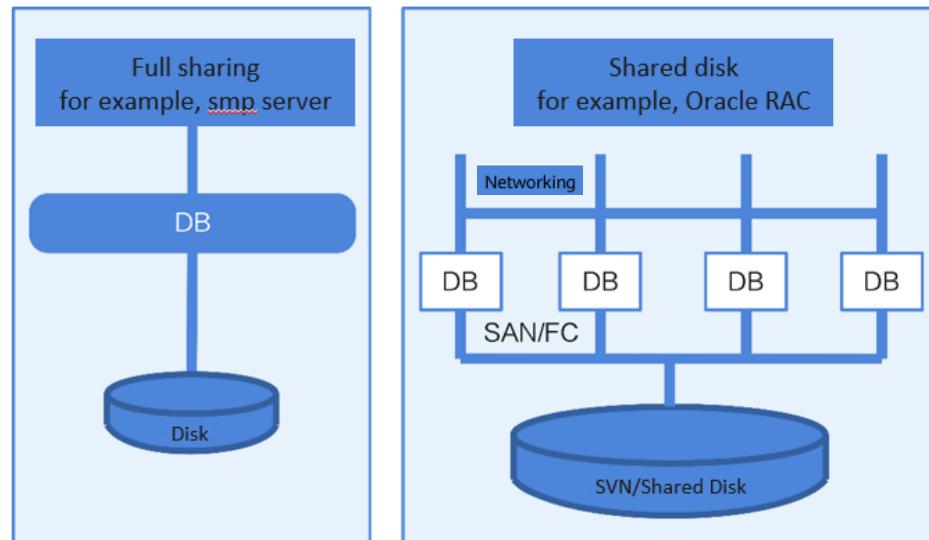
High Scalability - Shared-Nothing Architecture

- Shared-Nothing architecture
 - Each node (processing unit) in the cluster has its own independent CPU, memory, and storage, and does not share resources.
 - Each node (processing unit) processes its own local data. The processing result can be summarized to the upper layer or transferred between nodes through communication protocols.
 - Nodes are independent of each other and have strong scalability. The entire cluster has powerful parallel processing capabilities.



High Scalability - Advantages of the Shared-Nothing Architecture

Architecture comparison:

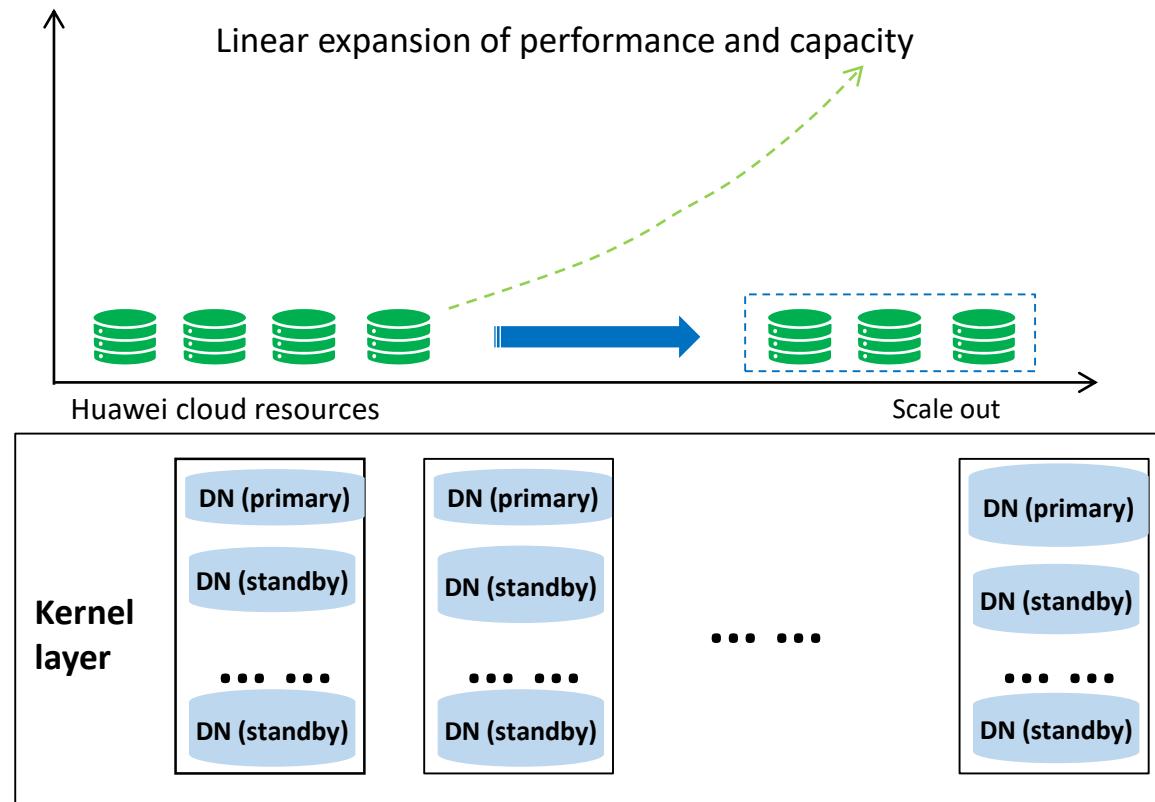


Advantages of the Shared-Nothing architecture:

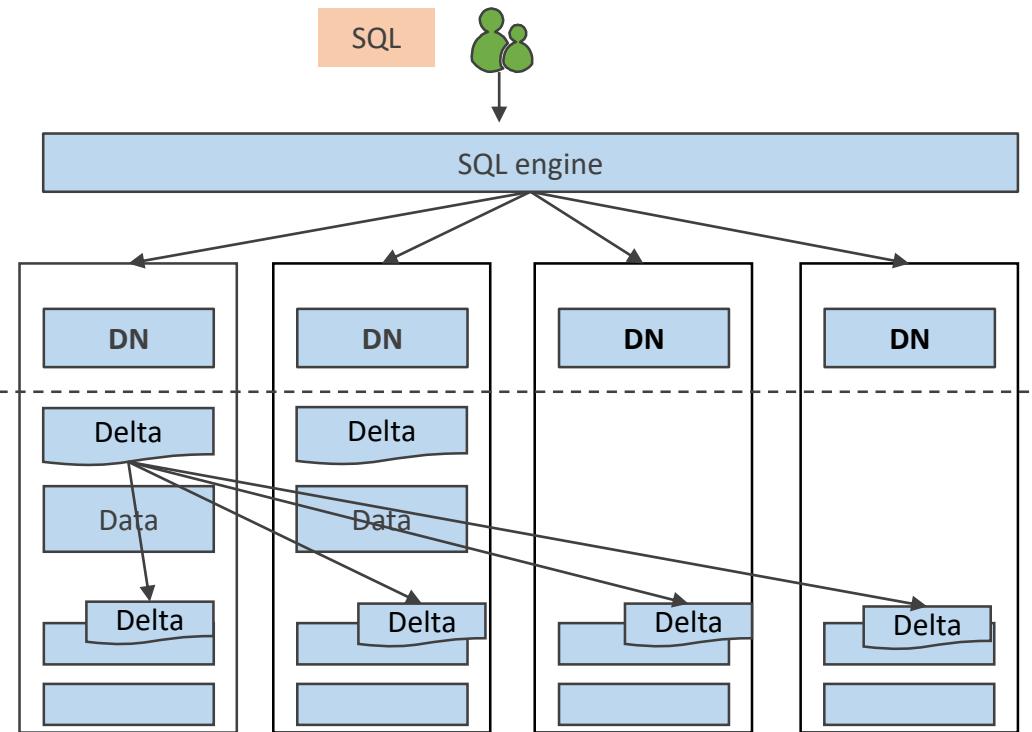
- **High scalability**
 - Provides high scalability for BI and data analysis to process high-concurrency and large-data-amount computing.
 - Supports the parallel processing mechanism.
- **Automated internal parallel processing**
 - Allows you to load and access data as you would in a common database.
 - Distributes data on all parallel nodes.
 - Each node processes only partial data.
- **Optimal I/O processing**
 - All nodes process data in parallel.
 - Nodes share nothing with each other and have no I/O conflicts.
- **Performance expansion by adding nodes**
 - The storage, query, and loading performance can be improved by adding nodes.

High Scalability

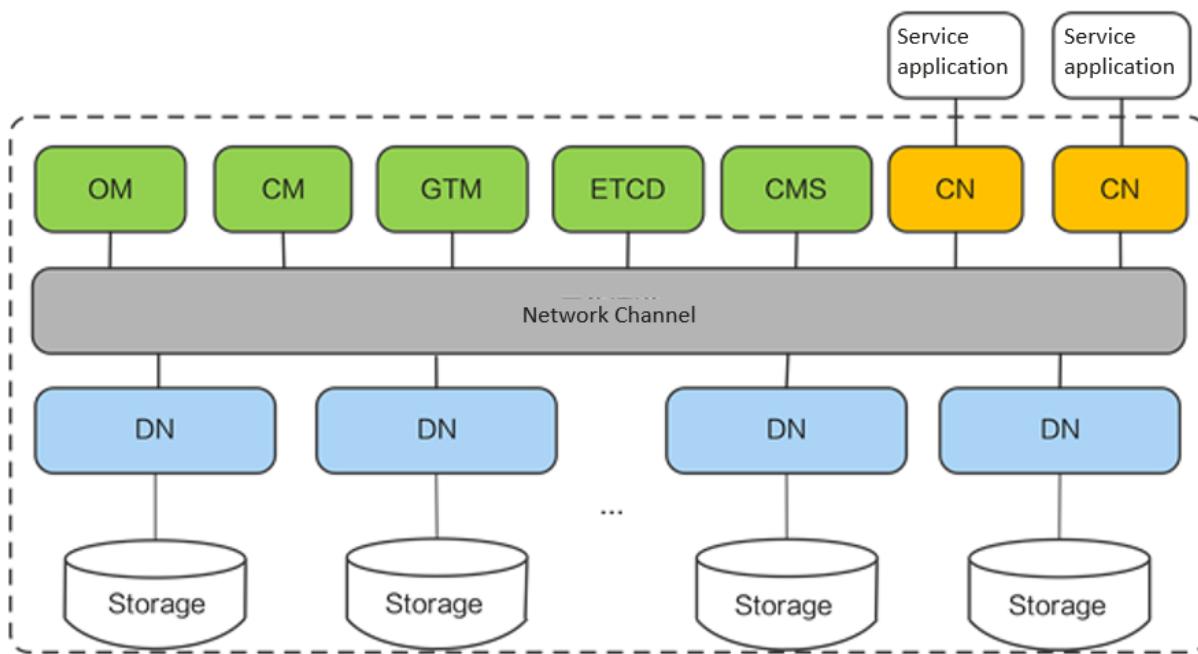
Performance and capacity



Online scale-out



GaussDB(for openGauss) Distributed Architecture - Component Functions



Roles perceived by the management plane:

- DN
- CN
- GMT
- ETCD
- CMS

Name	Description
OM	Provides management interfaces and tools for routine maintenance and configuration management of the cluster.
CM	Manages and monitors the running status of functional units and physical resources in the distributed system, ensuring stable running of the entire system.
GTM	Generates and maintains globally unique information, such as the global transaction ID, transaction snapshot, timestamp, and sequence.
CN	Receives access requests from the application layer, and returns the execution result to the client. The CN breaks down tasks and distributes task fragments to different DNs for parallel processing.
DN	Stores service data by column or row or in the hybrid mode, executes data query tasks, and returns execution results to CNs.
ETCD	Used for sharing configurations and discovering services (service registration and search).
Storage	Functions as the server's local storage resources to store data permanently.

Contents

1. Introduction to GaussDB(for openGauss)
- 2. Enterprise-Level Features of GaussDB(for openGauss)**
3. Comprehensive Tools and Service-oriented Capabilities
4. Application Scenarios and Cases

Enterprise-Level Feature - Distributed Storage

- As a distributed database, GaussDB(for openGauss) provides two-layer distributed storage capabilities.
 - For distributed storage of data between shards, both hash and replication distribution are supported. Hash distribution is mainly applied to user tables with a large amount of data. Hash calculation is performed on one or more distribution keys specified by a user, and data within the same user table is distributed and stored in different shards, thereby increasing the total data volume that can be supported by an entire database. In addition, distributed parallel processing capabilities and pruning processing capabilities are provided. Replication distribution is mainly used for user tables with a small amount of data. All of the replication distribution table's data is stored in each shard, improving the performance of distributed multi-table associated query.
 - Distributed storage of data in a shard: For each shard, quorum-based distributed multi-copy storage is supported to ensure high reliability and availability of the database. In addition, functions such as automatic primary/standby switchover, AZ switchover, and forcible promotion to the primary are provided, ensuring that the RPO and RTO are stable and meet expectations.

Key Feature - Data Partitioning (1)

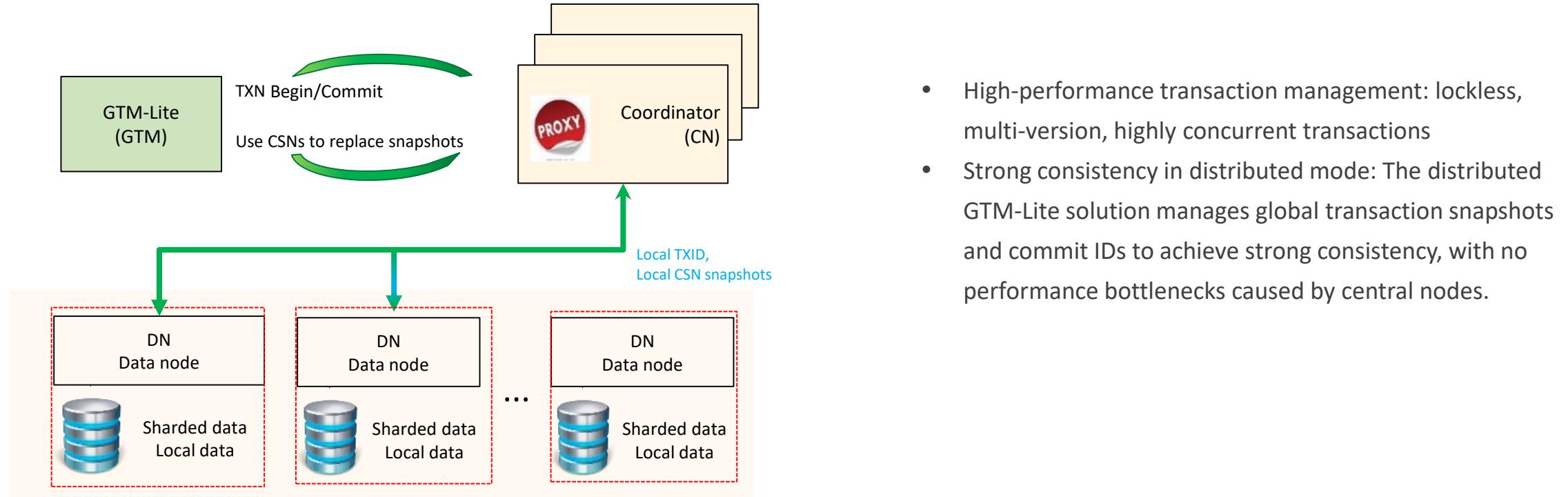
- Data partitioning is a common function among database products. In GaussDB(for openGauss), data is partitioned horizontally with a user-specified policy. This operation splits a table into multiple partitions that do not overlap.
- GaussDB(for openGauss) supports range partitioning. Records from one or more columns to be inserted into a table are divided into multiple ranges. A partition for each range is created to store data, and partition ranges do not overlap. If you specify the **PARTITION** parameter when running the **CREATE TABLE** statement, data in the table will be partitioned.

Key Feature - Data Partitioning (2)

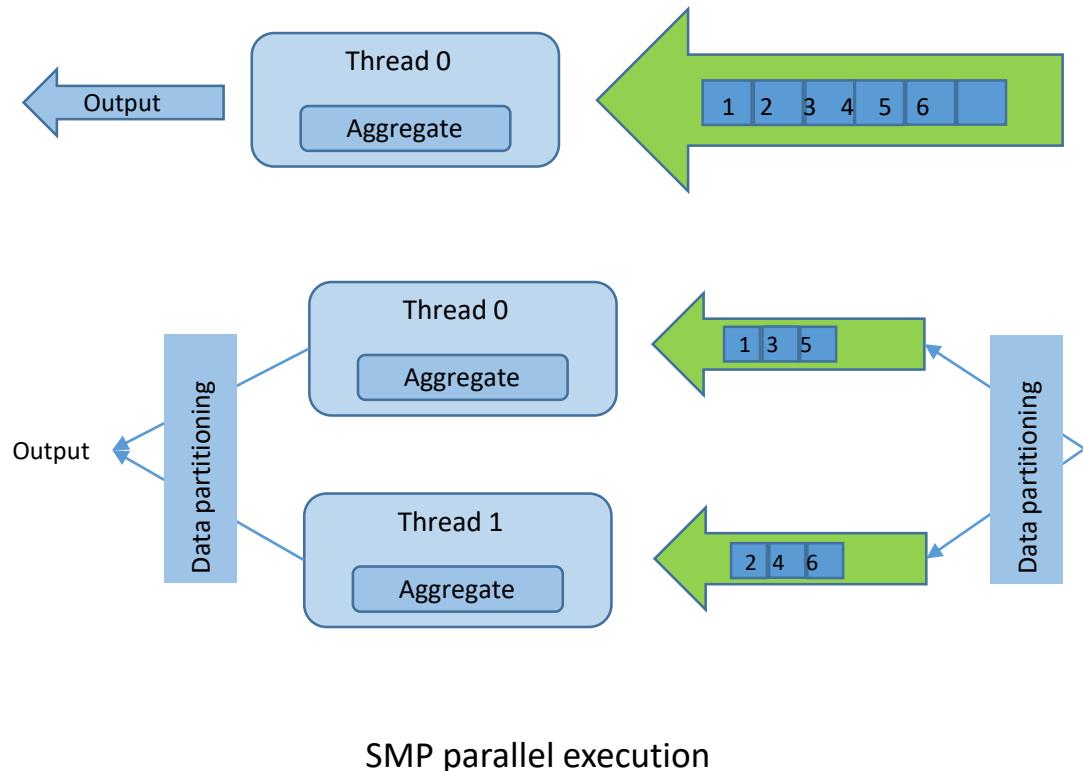
- Advantages of data partitioning:
 - Improved manageability: Tables and indexes are divided into smaller and more manageable units. This helps database administrators manage data based on partitions, allowing maintenance to be performed for specific parts of a table.
 - Better deletion performance: Deleting a partition is more efficient and faster than deleting rows.
 - Stronger query performance: You can restrict the volume of data to be checked or operated on for stronger performance.

Distributed Transaction Processing Performance: GTM-Lite Technology

- GTM-Lite technology offers high-performance transaction processing and ensures strong global consistency, eliminating the performance bottlenecks associated with single GTM.

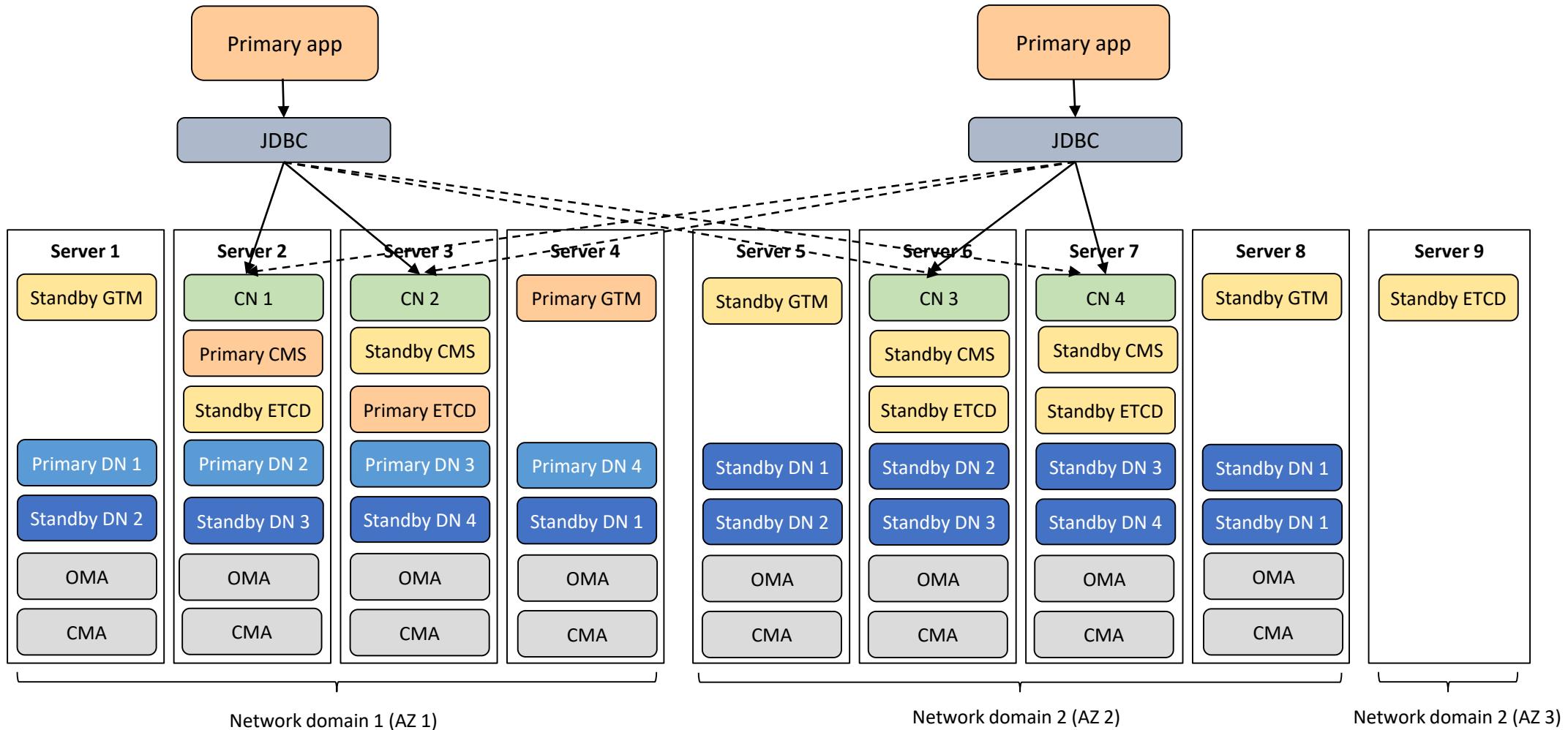


SMP Technology



- The SMP technology of GaussDB(for openGauss) uses a computer's multi-core CPU architecture to implement multi-thread parallel computing, fully utilizing CPU resources to improve query performance.
 - SMP refers to the parallel execution of tasks at the **thread** level. Theoretically, the number of subtasks that can be concurrently executed can reach the upper limit of the number of cores on the physical server.
 - SMP parallel threads are in the same process, and **data can be exchanged directly through memory**, without occupying network connection and bandwidth. This reduces the impact of network factors that restrict the performance improvement of the MPP system.
 - As the parallel subtasks **do not need to be attached to another background worker thread** after being started, the utilization rate of system computing resources can be increased effectively.

Deployment Mode - Three AZs with Four Replicas



Physical Backup

- GaussDB(for openGauss) provides a physical backup function capable of backing up data from the entire cluster to Object Storage Service (OBS) in the internal format of databases, as well as restoring such data in homogeneous databases. Physical backup uses distributed parallel technology to physically back up data files of each DN, offering high backup and restoration performance. In addition, it also provides such advanced functions as compression, flow control, and resumable backup.
- Physical backup is classified as either full or incremental backup. Full backup includes the full data of the database at the backup time point, requires an extended timeframe (in direct proportion to the total data volume of the database), and enables a complete database to be restored. In contrast, an incremental backup involves only incremental data modified after a specified time point. It requires a shorter timeframe (in direct proportion to the incremental data volume and not related to the total data volume). However, a complete database can be restored only after both the incremental backup and full backup are performed. You can create a backup policy every week.
 - On Monday, perform a full backup of the database.
 - On Tuesday, perform an incremental backup based on Monday's full backup.
 - On Wednesday, perform an incremental backup based on Tuesday's incremental backup.
 - ...
 - On Sunday, perform an incremental backup based on Saturday's incremental backup.

Logical Backup

- GaussDB(for openGauss) provides logical backup capabilities to back up data in user tables to local disk files and OBS in text or CSV format, and restore the data in homogeneous or heterogeneous databases. Logical backup uses the distributed parallel technology to extract user table records to be backed up from each DN in streaming mode, providing high backup and restoration performance.

Contents

1. Introduction to GaussDB(for openGauss)
2. Enterprise-Level Features of GaussDB(for openGauss)
- 3. Comprehensive Tools and Service-oriented Capabilities**
4. Application Scenarios and Cases

DAS

Data Admin Service (DAS) is a **one-stop cloud database management platform** that allows you to manage databases on a web-based console. It offers **database development, O&M, intelligent diagnosis, and enterprise-level DevOps** to facilitate your cloud database **usage** and **maintenance**.

Standard edition

Intended for developers

Provides a best-in-class experience. You no longer need to install clients locally for a visualized operation experience. Diverse database development functions are available, including online editing and intelligent prompts for SQL input.

Cloud DBA

Intended for DBAs and O&M personnel

Provides a wide range of functions, covering instance performance data, slow SQL analysis, SQL explorer, real-time performance analysis and diagnosis, as well as historical running data analysis, helping you to quickly locate every issue encountered during the lifetime of the database and avoid potential risks.

Enterprise edition

Intended for enterprise users

Offers a DevOps database product with a focus on database access, operation security, and communication of programmers, DBAs, and enterprise managers. It combines automatic identification of modification risks and the permission approval process to ensure data change security and improved development efficiency.

DRS

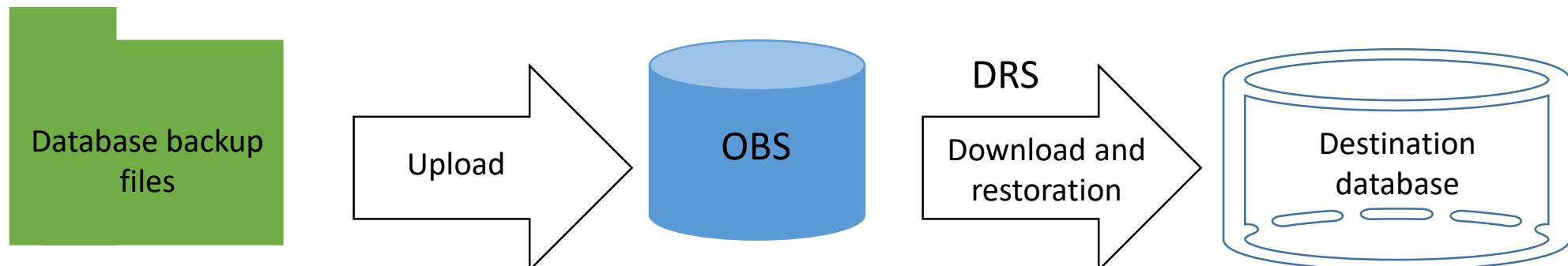
- Data Replication Service (DRS) is a stable, efficient, and easy-to-use cloud service for real-time database migration and synchronization.
- DRS simplifies data transmission between databases and reduces data movement costs. You can use DRS to migrate data between databases quickly in order to meet your service requirements.
- DRS provides multiple functions, including real-time migration, backup migration, real-time synchronization, data subscription, and real-time DR.

DRS Service Module - Real-Time Migration

- Real-time migration allows you to directly migrate data from the source database to the destination database on the condition that they are connected and the source database instances, destination database instances, and migration objects are configured.
- Real-time migration supports multiple types of networks, such as public networks, VPCs, VPNs, and Direct Connect. With these networks, migration can be performed between different cloud platforms, from on-premise databases to cloud databases, or on the cloud across regions.
- DRS supports incremental migration, which ensures your service continuity while minimizing the service downtime and migration impact. In this way, databases can be smoothly migrated to the cloud, and all database objects can be migrated.

DRS Service Module - Backup Migration

- It often becomes necessary to hide one of your database's real IP addresses for the sake of security. Migrating data through direct connections is an option, but can be costly. DRS supports backup migration, which allows you to export data from your source database for backup and upload the backup files to OBS. You can then restore the backup files to the destination database to complete the migration. Using this method, data migration can be achieved without exposing your source databases.
- Common scenario: on-premise to cloud migration. Currently, only SQL Server can be migrated to the cloud.
- Without connecting to your sources databases, DRS can help you complete data migration.

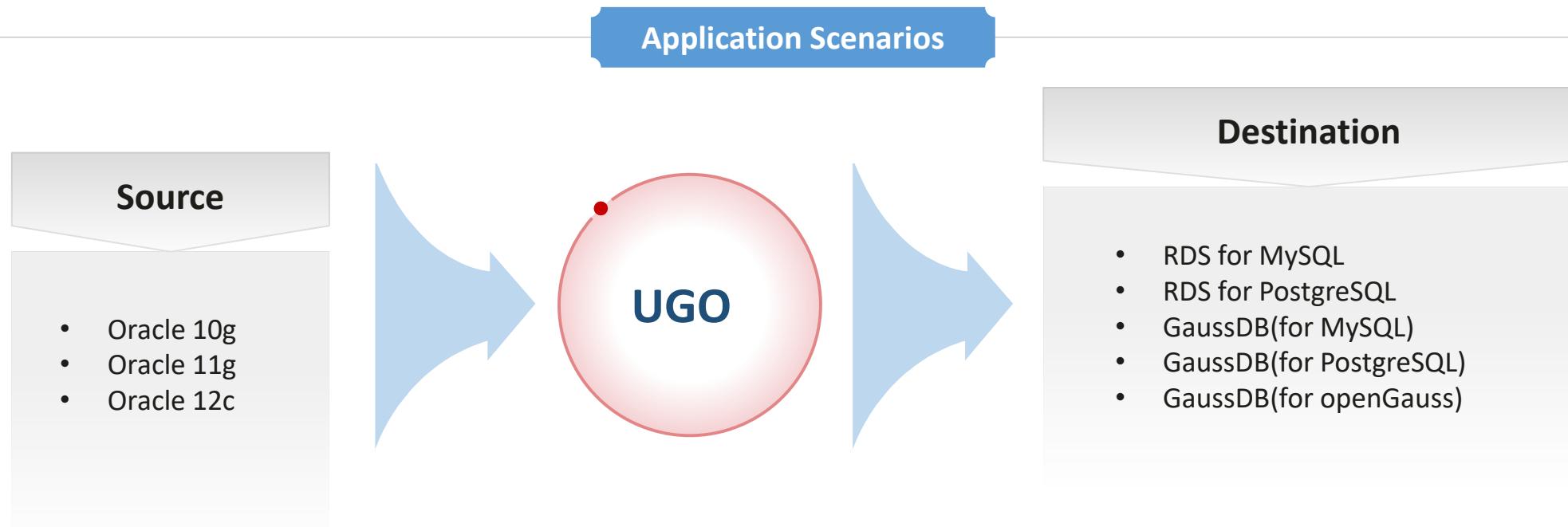


DRS Service Module – Real-Time Synchronization

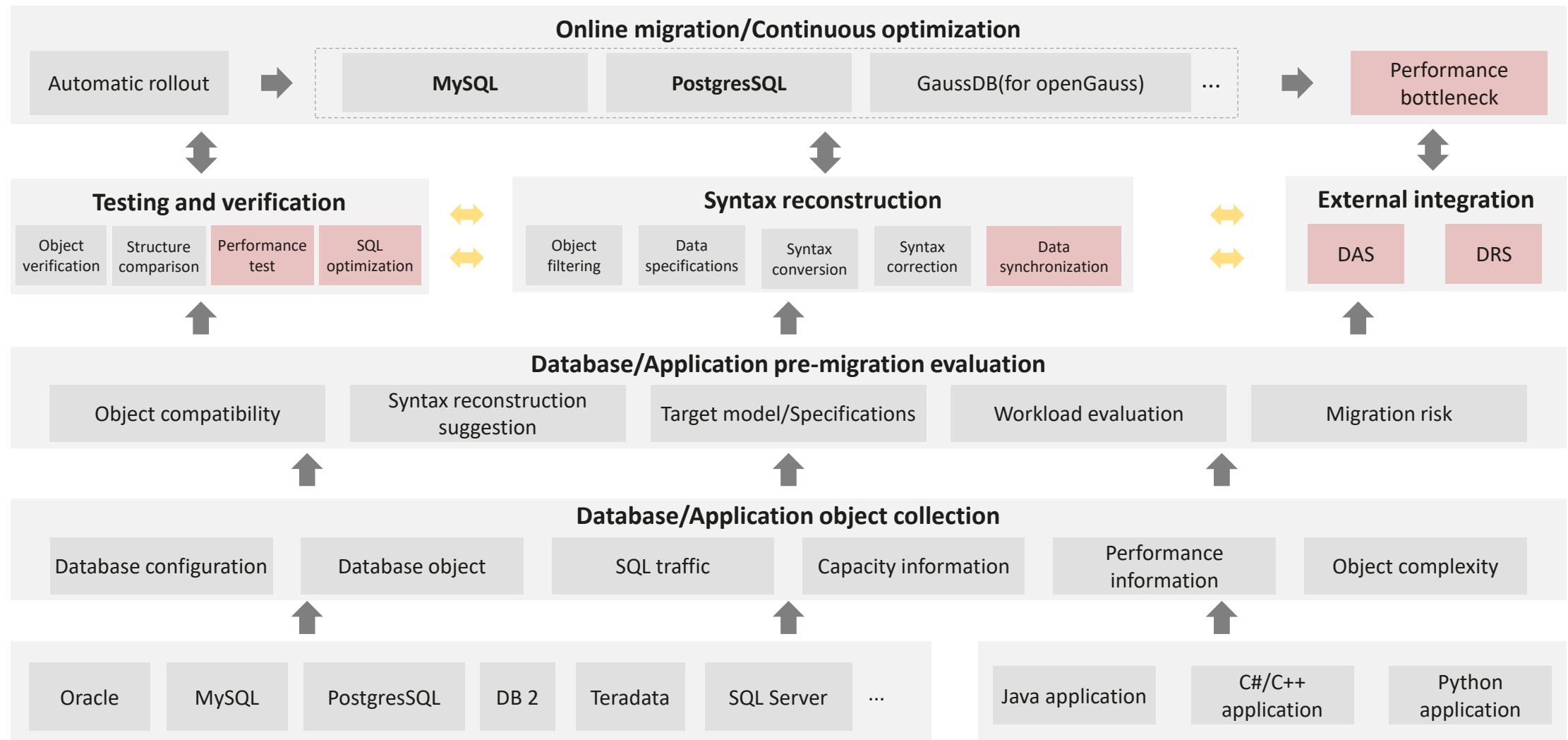
- Real-time synchronization refers to the real-time replication of key service data from the source database to the destination database through DRS, while ensuring consistency of the data.
- Different from data migration, which involves moving your overall database from one platform to another, synchronization refers to the continuous flow of data between different services.
- Common scenarios: real-time analysis, report system, and data warehouse environment
- DRS can satisfy various requirements, such as many-to-one, one-to-many synchronization, dynamic addition and deletion of synchronization tables, and synchronization between tables with different names.

UGO Service

Database and Application Migration UGO, referred to as UGO, is a professional tool that focuses on **heterogeneous database object migration** and **application migration**. As such, it enables automatic migration of mainstream commercial databases to HUAWEI CLOUD databases. Data is **easily migrated to the cloud**, and **databases can be switched with just a few clicks**.



UGO Product Migration Solution



UGO Core Functions

**Source database
profiling**

**Destination database
selection and
specification
recommendation**

**Destination database
compatibility analysis**

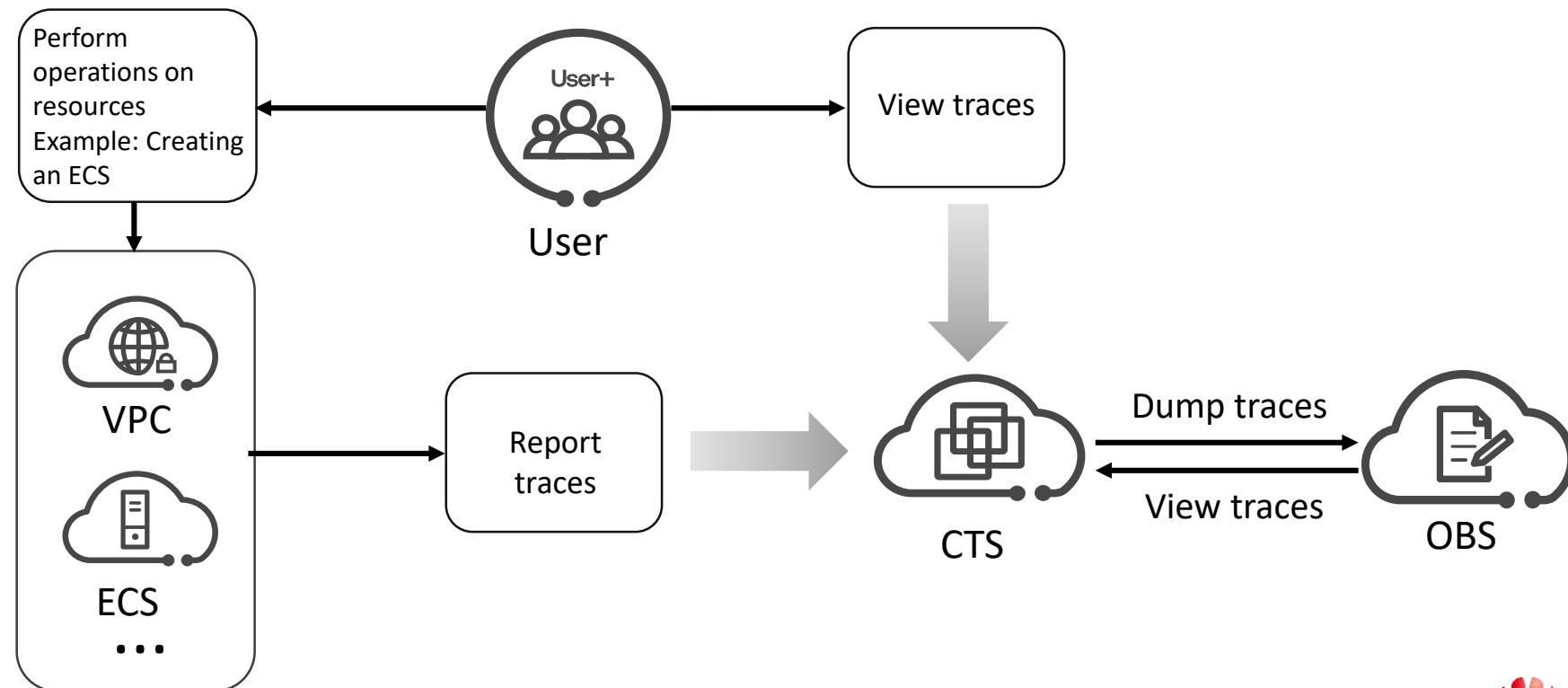
**Migration workload
evaluation**

**Database structure
migration**

**Application SQL
migration**

CTS

- Cloud Trace Service (CTS) is a log audit service that is available on HUAWEI CLOUD, and has been designed to offer enhanced cloud security. It allows you to collect, store, and query cloud resource operation records, and then use these records for security analysis, compliance auditing, resource tracking, and fault locating.

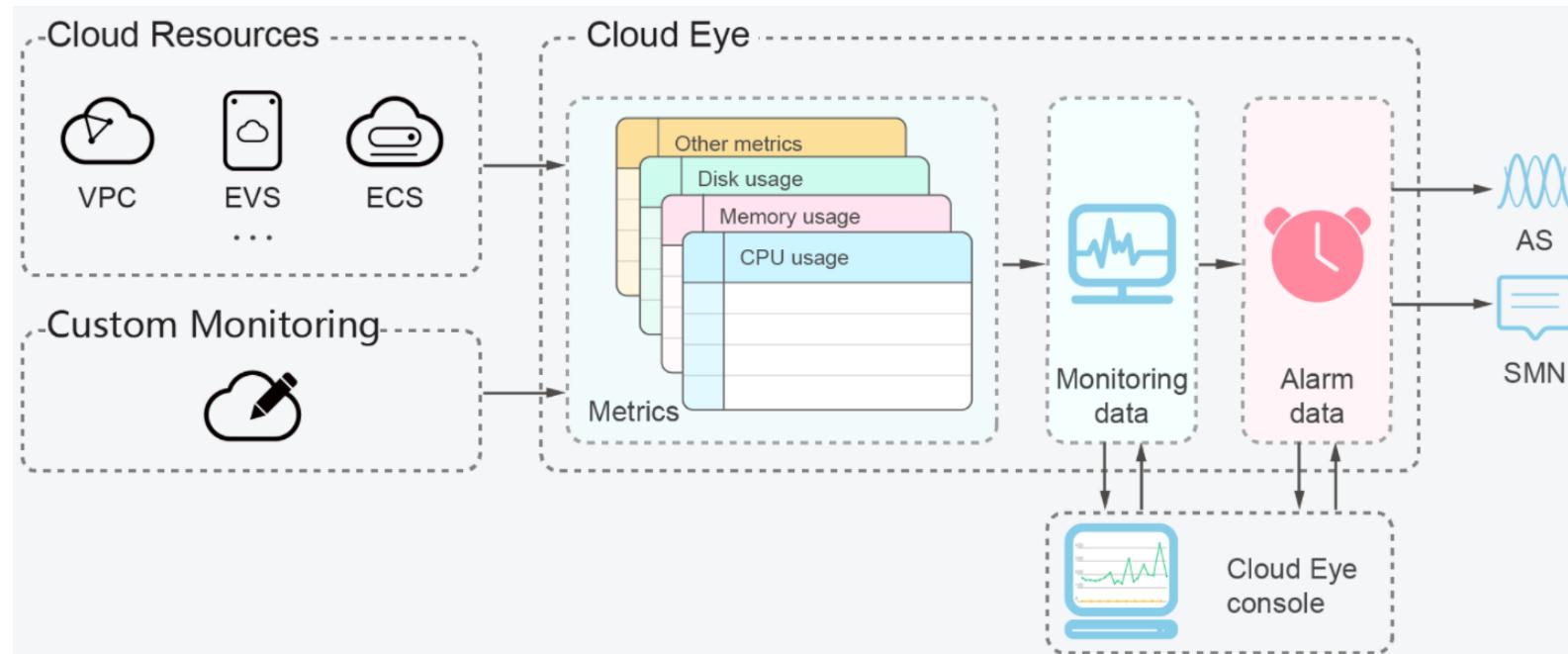


CTS Functions

- Trace recording: CTS records operations performed on the management console or by calling APIs, as well as operations triggered by each interconnected service.
- Trace query: Operation records from the last seven days can be queried on the management console from multiple dimensions, such as trace type, trace source, resource type, filter, operator, and trace level.
- Trace dumping: Traces are delivered to OBS buckets on a regular basis for long-term storage. Through this process, traces are compressed into trace files by service.
- Trace file encryption: Trace files are encrypted using keys provided by Key Management Service (KMS) during dumping.

CES (1)

- Cloud Eye Service (CES) is a comprehensive monitoring platform for resources such as the ECS and bandwidth. You can use CES to monitor the utilization of service resources, track the running status of cloud services, configure alarm rules and notifications, and quickly respond to resource changes.



CES (2)

- CES provides the following functions: automatic monitoring, server monitoring, real-time notification, monitoring panel, OBS dumping, resource grouping, log monitoring, and trace monitoring.
- GaussDB(openGauss) can use CES to monitor the operating status of the system. By monitoring system resource usage when the database is running, you can identify periods of high resource usage. To optimize database performance, you can analyze which SQL statements may have had performance issues by checking error logs or slow logs.

DBSS

- Database Security Service (DBSS) leverages Huawei's 30 years of experience in database security practices. It offers two subservices, database security audit and database protection, which deliver such functions as data breach prevention, database firewall, and database security audit to protect your databases and assets on the cloud.
- Database security audit
 - Database security audit provides an audit function in bypass mode, enabling the system to generate real-time alarms for risky operations and attack behaviors. In addition, database security audit generates compliance reports that meet data security standards. In this way, it locates internal violations and improper operations, detects and blocks external intrusions, and protects data assets. Functions such as user behavior detection and auditing, multi-dimensional lead analysis, real-time alarms, and reports are all supported.

Database Security Audit (1)

- User behavior detection and audit
 - Access operations at the application layer and database layer can be associated to help customers trace the identities and behaviors of users.
- Multi-dimensional lead analysis
 - Risk leads: This function can analyze SQL behaviors such as SQL injection, blacklist statements, and authorization violation at high, medium, and low risk levels.
 - Session leads: This function supports analysis from multiple perspectives, including time, user, IP address, and client.
 - Detailed statement leads: This function supports searching by user, client IP address, access time, operation object, and operation type.

Database Security Audit (2)

- Real-time alarms for abnormal operations, SQL injection, as well as blacklist and whitelist hits
 - Abnormal behavior detection: This function monitors access behaviors based on multiple fine-grained factors such as the client IP address, database IP address, database account, and risk severity.
 - SQL injection: This function provides a systematic SQL injection database and SQL injection description based on regular expressions or syntax abstraction, and generates alarms when detecting database exceptions.
 - Blacklist and whitelist: SQL statements for accessing the system are described accurately and abstractly, and alarms are generated in real time when the SQL statements appear.

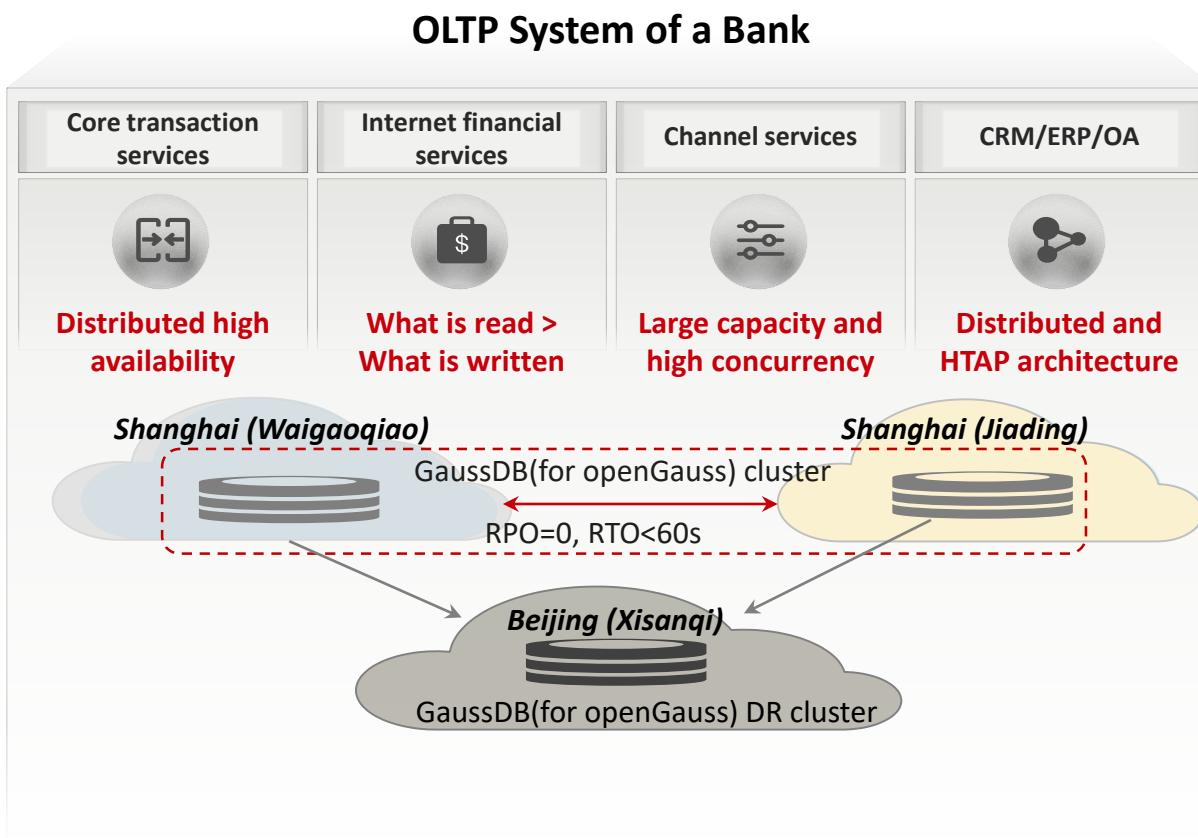
Database Security Audit (3)

- Fine-grained reports for various abnormal behaviors:
 - Session behavior: login failure and session analysis reports
 - SQL behavior: new SQL, SQL statement execution history, and SQL failure reports
 - Risky behavior: alarm, notification, SQL injection, and batch data access behavior reports
 - Compliance report: compliance reports that meet data security standards (such as Sarbanes-Oxley)

Contents

1. Introduction to GaussDB(for openGauss)
2. Enterprise-Level Features of GaussDB(for openGauss)
3. Comprehensive Tools and Service-oriented Capabilities
- 4. Application Scenarios and Cases**

HUAWEI CLOUD GaussDB Supports In-House Distributed Reconstruction of a Bank's Core Service System (1)



Challenges

- **Capacity bottleneck:** Rapid development of Internet financial services, poor scalability of traditional centralized databases, and urgent need for distributed reconstruction have caused bottlenecks.
- **High availability in finance:** Core financial services are migrated to an open platform. Benchmarking DB2 is required to provide high availability in intra-city and active-active DR and 3DC geo-redundant DR in finance..

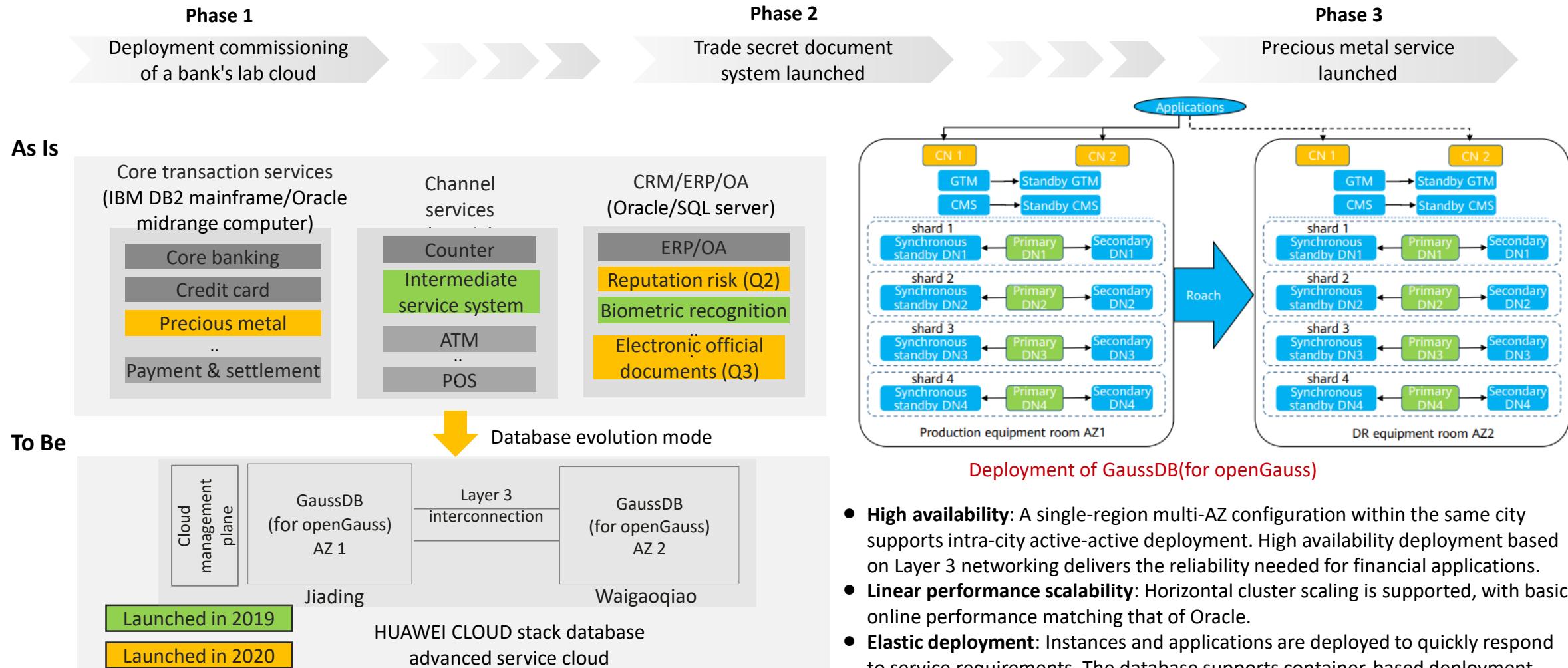
Solutions

- **Distributed high scalability:** Distributed expansion capabilities and ACID protection for distributed transactions are provided.
- **3DC geo-redundant DR:** A single cluster is deployed across AZs in intra-city active-active mode to achieve an RPO of zero and an RTO of less than 60s. Dual-cluster DR is deployed cross regions.

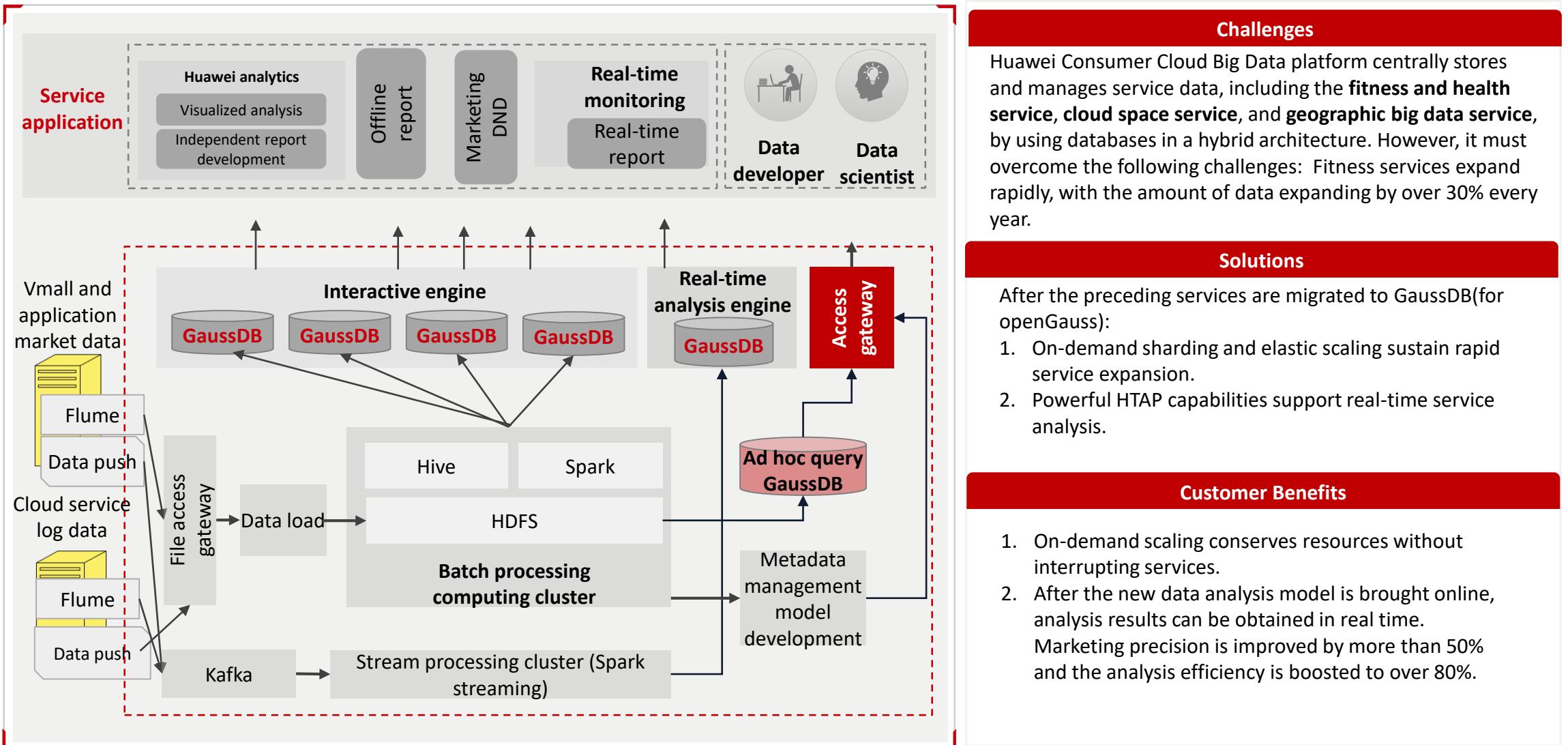
Customer Benefits

- **Large capacity and high scalability:** Terabyte- or petabyte-level data for a single database and online scale-out are supported with no sharding, simplifying application development.
- **Financial high availability:** Intra-city active-active deployment enables services from two DCs within the same city to be accessed at the same time. If one DC becomes faulty, services can be recovered within seconds.

HUAWEI CLOUD GaussDB Supports In-House Distributed Reconstruction of a Bank's Core Service System (2)



GaussDB Helps Huawei Consumer Cloud Implement Smart Service Operation



Challenges

Huawei Consumer Cloud Big Data platform centrally stores and manages service data, including the **fitness and health service**, **cloud space service**, and **geographic big data service**, by using databases in a hybrid architecture. However, it must overcome the following challenges: Fitness services expand rapidly, with the amount of data expanding by over 30% every year.

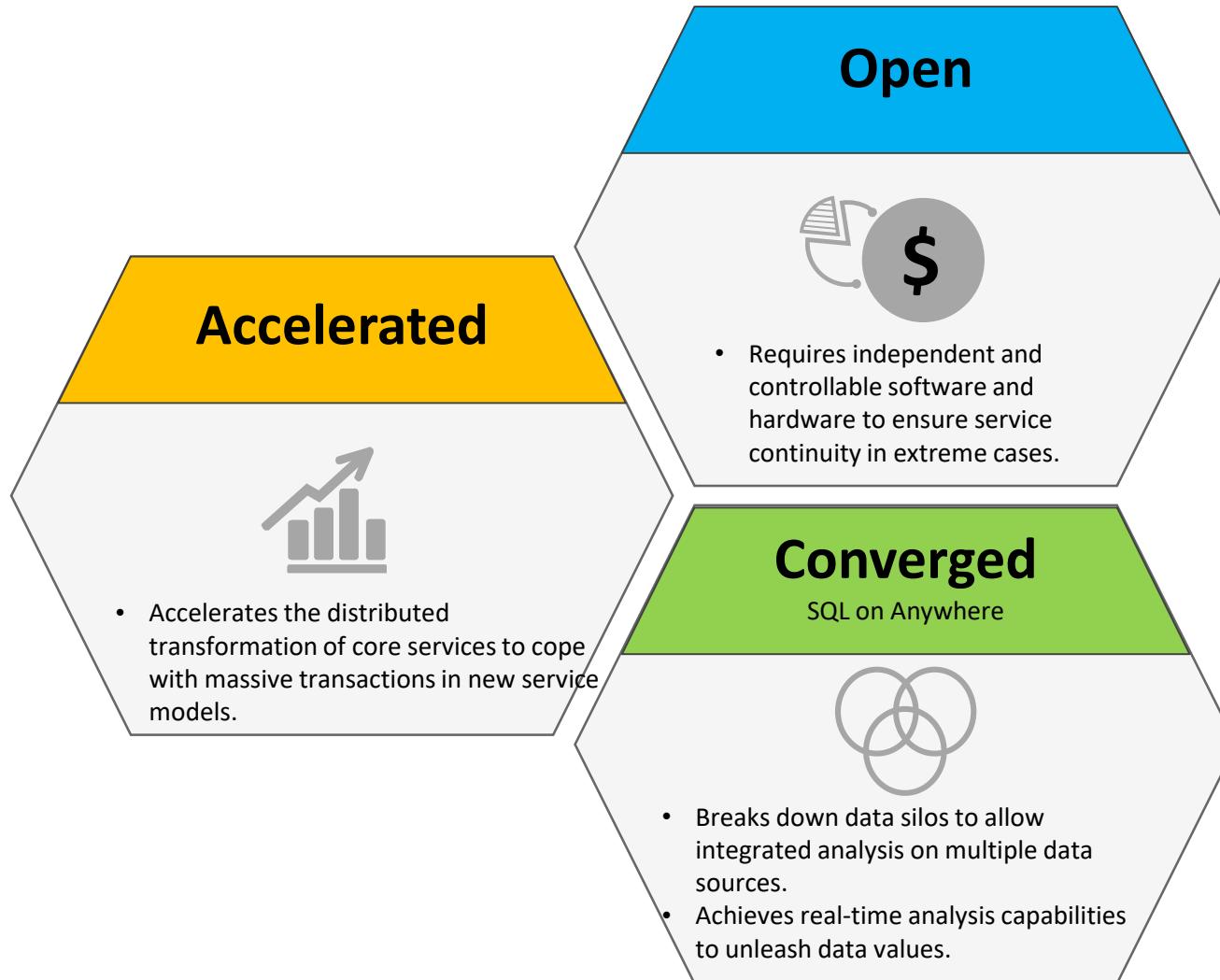
Solutions

- After the preceding services are migrated to GaussDB(for openGauss):
1. On-demand sharding and elastic scaling sustain rapid service expansion.
 2. Powerful HTAP capabilities support real-time service analysis.

Customer Benefits

1. On-demand scaling conserves resources without interrupting services.
2. After the new data analysis model is brought online, analysis results can be obtained in real time. Marketing precision is improved by more than 50% and the analysis efficiency is boosted to over 80%.

GaussDB(for openGauss) Enables Smooth Transformation of Customer Services in the Financial Industry



Service Requirements and Challenges

Customers in the financial industry in China require the following:

1. Independent and controllable core services
2. Transformation to a distributed transaction system

Solutions

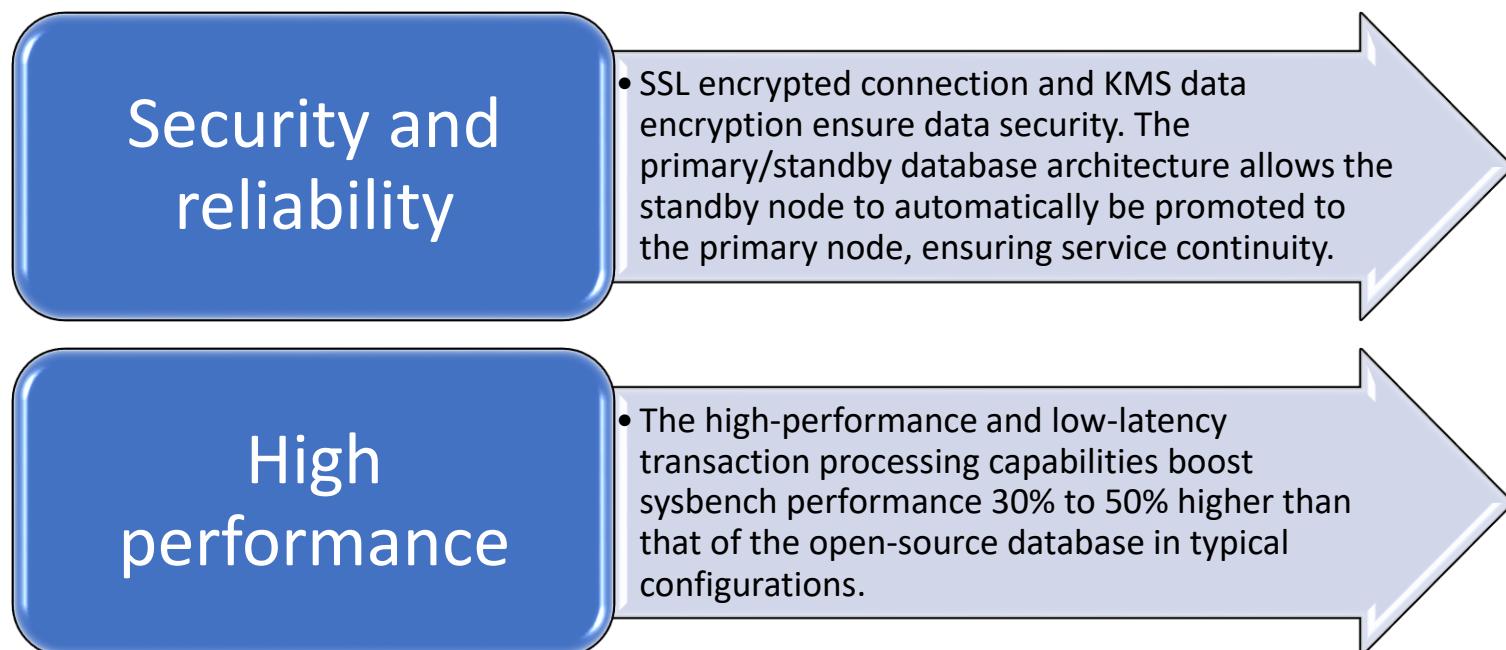
1. HUAWEI CLOUD provides a full-stack, independent, and self-controllable solution for software and hardware.
2. Various migration tools are used to efficiently and cost-effectively complete online service migration and distributed reconstruction.
3. Powerful HTAP capabilities, desirable storage project compatibility, and real-time service analysis are supported.

Customer Benefits

1. In-house independent and controllable capabilities are achieved.
2. A solid foundation is laid for continuous service expansion.
3. Data values are unleashed more effectively.

Financial Internet Transaction

- GaussDB(for openGauss) is applicable to the Internet transaction systems of small- and medium-sized banks, such as mobile apps and websites. It is compatible with mainstream commercial database ecosystems, features high performance, security, and reliability, and is recommended to be deployed in active/standby mode.
- Advantages:



Quiz

1. (Single-answer) In the GaussDB(for openGauss) distributed architecture, which of the following components is used for sharing configuration and service discovery (service registration and search)?
 - A. CN
 - B. DN
 - C. GTM
 - D. ETCD
2. (Short-answer) What are the core functions of UGO for database and application migration?

Summary

- This chapter describes the positioning of GaussDB(for openGauss), as well as its key features, such as high performance, high availability, high security, and high scalability.

Recommendations

- HUAWEI CLOUD:
 - <https://www.huaweicloud.com/>
- openGauss help document:
 - <https://opengauss.org/en/docs/2.0.0/docs/Quickstart/Quickstart.html>
- openGauss code repository:
 - <https://gitee.com/opengauss>

Acronyms and Abbreviations

Acronym and Abbreviation	Full Spelling
AZ	Availability Zone
CBO	Rule-Based Optimization
CM	Cluster Manager
CN	Coordinator Node
CTS	Cloud Trace Service
DAS	Data Admin Service
DBSS	Database Security Service
DN	Data Node
DRS	Data Replication Service
GTM	Global Transaction Manager
OM	Operation Manager
RBO	Rule-Based Optimization
SMP	Symmetric Multi-Processing
UGO	Database and Application Migration UGO

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and
organization for a fully connected,
intelligent world.

Copyright©2022 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

