ALIBABA CLOUD

Alibaba Cloud

AnalyticDB for PostgreSQL
Product Introduction

Document Version: 20220712

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Overview

is a massively parallel processing (MPP) data warehousing service that is designed to analyze large volumes of data online.

is developed based on the open source Greenplum Database project and is enhanced with in-depth extensions by Alibaba Cloud. AnalyticDB for PostgreSQL is compatible with the ANSI SQL 2003 syntax and the PostgreSQL and Oracle database ecosystems. AnalyticDB for PostgreSQL also supports row store and column store. AnalyticDB for PostgreSQL processes petabytes of data offline at a high performance level and supports highly concurrent online queries. This way, AnalyticDB for PostgreSQL can provide a competitive data warehousing solution for a variety of industries.

Features

- Adaptable to variable workloads without optimization required.

  AnalyticDB for PostgreSQL is fully compatible with SQL 2003 syntax and partially compatible with Oracle syntax. This service also supports PL/SQL stored procedures. AnalyticDB for PostgreSQL offers next-generation query optimizers to eliminate the need to optimize complex SQL statements.

- Analyzes petabytes of data within seconds.

  AnalyticDB for PostgreSQL uses an MPP scale-out architecture to respond to queries for petabytes of data within seconds. AnalyticDB for PostgreSQL supports vector computing and intelligent column store indexing. The performance of AnalyticDB for PostgreSQL is about ten times better than that of a traditional database engine.

- Provides high availability and always-on connectivity.

  AnalyticDB for PostgreSQL supports distributed transactions, redundancy for all nodes and data, automatic monitoring and failover for hardware faults, and atomicity, consistency, isolation, durability (ACID).

- Compatible with a wide variety of ecosystems.

  AnalyticDB for PostgreSQL supports mainstream business intelligence (BI) and extract, transform, and load (ETL) tools. For example, AnalyticDB for PostgreSQL is integrated with the PostGIS extension to analyze geographic data and with the MADlib library to provide more than 300 built-in machine learning algorithms.

- Enables data synchronization.

  AnalyticDB for PostgreSQL can synchronize data from a variety of data sources by using tools such as Data Transmission Service (DTS) and DataWorks. AnalyticDB for PostgreSQL supports highly parallel access to Object Storage Service (OSS) and Data Lake Analytics (DLA).

Service architecture

uses the MPP architecture. In this architecture, an instance is composed of multiple compute nodes. The storage type can be ultra disk or enhanced SSD (ESSD). Computing is decoupled from storage. You can add compute nodes to an instance or scale up its storage capacity and maintain a stable response time. Each instance is composed of the following components:

- Coordinator node
  - Receives query requests and determines distributed query plans.
- Compute node
  - Provides MPP.

- Stores data partitions on two compute nodes.
- Automatically backs up data to OSS on a regular basis.

As of February 8, 2021, you are allowed to configure multiple coordinator nodes in . Greenplum Database does not support this feature. You can add multiple coordinator nodes to an instance to push beyond the limits of the original architecture in which an instance has a single coordinator node. If compute nodes permit, the number of connections and the I/O capabilities can linearly increase with the number of coordinator nodes. This enhances the overall performance of the system to better meet the requirements of business scenarios such as real-time data warehousing and hybrid transaction/analytical processing (HTAP). For an instance that has multiple coordinator nodes, AnalyticDB for PostgreSQL provides instance endpoints in addition to the primary coordinator node endpoints. For more information, see Endpoints of an instance and its primary coordinator node.

References

- To learn more about , you can go to the Alibaba Cloud Yunqi community and join the AnalyticDB for PostgreSQL technical support group on DingTalk. For more information, see Get started with AnalyticDB for PostgreSQL.
- If you require technical assistance, submit a ticket.
- For information about the open source Greenplum Database project, visit the Greenplum Database official website.

Precautions

is developed based on the open source Greenplum Database project and is enhanced with in-depth extensions by Alibaba Cloud. In , some features of GreenPlum Database such as triggers are prohibited based on in-depth understanding and maintenance experience of the team on Greenplum Database. For more information about limits of AnalyticDB for PostgreSQL, see Limits.

2.Scenarios

This topic describes the common scenarios and benefits of AnalyticDB for PostgreSQL.

Scenarios

- **Data warehousing service**

  You can use Data Transmission Service (DTS) or Data Integration to synchronize large amounts of data from Alibaba Cloud database services such as ApsaraDB RDS and PolarDB or self-managed databases to AnalyticDB for PostgreSQL. AnalyticDB for PostgreSQL supports Extract, Transform, and Load (ETL) operations on large amounts of data. You can also use DataWorks to schedule these tasks. AnalyticDB for PostgreSQL also provides high-performance online analysis capabilities. You can use Business Intelligence (BI) tools such as Quick BI, DataV, Tableau, and FineReport to query real-time data that is stored in AnalyticDB for PostgreSQL and present the data in reports.

- **Big data analytics platform**

  You can use Data Integration or Object Storage Service (OSS) to import large amounts of data from MaxCompute, Hadoop, and Spark to AnalyticDB for PostgreSQL for high-performance analysis, processing, and online data exploration.

- **Data lake analytics**

  AnalyticDB for PostgreSQL allows you to use OSS foreign tables to directly query large amounts of data in OSS in parallel and build an Alibaba Cloud data lake analytics platform.

Benefits

AnalyticDB for PostgreSQL provides the following benefits for online analytical processing (OLAP) services:

- **ETL for offline data processing**

  AnalyticDB for PostgreSQL provides the following benefits that make it ideal for optimizing complex SQL queries as well as aggregating and analyzing large amounts of data:

  - Supports standard SQL syntax, OLAP window functions, and stored procedures.
  - Provides the ORCA query optimizer to enable complex queries without the need for tuning.
  - Uses the massively parallel processing (MPP) architecture that can analyze and process petabytes of data in seconds.
  - Provides column store-based high-performance scanning of large tables at a high compression ratio.

- **Online high-performance query**

  AnalyticDB for PostgreSQL provides the following benefits for real-time exploration, warehousing, and updating of data:

  - Allows you to process high-throughput data by performing operations such as INSERT, UPDATE, and DELETE.
  - Allows you to obtain results from point query within milliseconds based on row store and multiple indexes such as B-tree and bitmap indexes.
  - Supports distributed transactions, standard database isolation levels, and hybrid transaction/analytical processing (HTAP).

- **Multi-modal data analysis**

  AnalyticDB for PostgreSQL provides the following benefits for processing unstructured data from a variety of sources:

  - Supports the PostGIS extension for geographic data analysis and processing.
  - Uses the MADlib library of in-database machine learning algorithms to implement AI-native databases.
  - Provides high-performance retrieval and analysis of unstructured data such as images, audios, and text based on vector retrieval
  - Supports formats such as JSON and can analyze and process semi-structured data such as logs.

3.Instance specifications

This topic describes how to select instance specifications for .

Instance resource types

We recommend that you select the following instance resource types for :

- Elastic storage mode

  This instance resource type uses an integrated computing and storage architecture to provide comprehensive features. You can change compute node specifications, add compute nodes, and scale the storage capacity.

  When you purchase an instance in elastic storage mode, you must specify the **Edition**, **Compute Node Specifications**, **Nodes**, **Storage Disk Type**, and **Single Node Storage Capacity** parameters.

- Serverless mode

  This instance resource type uses an in-house decoupled computing and storage architecture to implement on-demand storage and scaling within seconds. It is ideal for scenarios where resource requirements fluctuate significantly.

  When you purchase an instance in Serverless mode, you must specify the **Edition**, **Compute Node Specifications**, and **Nodes** parameters.

The following tables describe these two instance resource types.

Elastic storage mode (recommended)

Edition	Node specification	Storage disk type	Suitable scenario
High-availability Edition	2 cores, 16 GB	Enhanced SSD (ESSD) Ultra disk	Proof of Concept (POC) testing. Individual learning and experience or testing of service features.
	4 cores, 32 GB		Balanced computing and storage scenarios. This specification is a choice for 60% of users.
	8 cores, 64 GB		Compute-intensive scenarios, where large amounts of complex data are analyzed or concurrently queried.
	16 core, 128 GB		Construction of enterprise-class platforms, where large amounts of enterprise core data are concurrently queried.
	2 cores, 8 GB		POC testing. Individual learning and experience or testing of service features.

Edition	Node specification	Storage disk type	Suitable scenario
High Performance (Basic Edition)	4 cores, 16 GB	ESSD	Balanced computing and storage scenarios for offline data analysis. **Notice** High Performance (Basic Edition) does not provide high availability. Proceed with caution.
	8 cores, 32 GB		
	16 cores, 64 GB		

Serverless mode (new)

Edition	Node specification	Storage disk type	Suitable scenario
High-availability Edition	4 cores, 16 GB	Shared storage	The new instance resource type can implement on-demand storage, data sharing, and node specification changes within seconds. It is suitable for the following scenarios: • Resource requirements fluctuate significantly. • New resource plans are not defined. • Business workloads are distinctly isolated.
	8 cores, 32 GB		

Capability comparison in different scenarios

The following table describes the typical data import scenarios.

Scenario	Elastic storage mode	Serverless mode
Data import from ApsaraDB RDS	Supported	Not supported
Data import from Realtime Compute for Apache Flink	Supported	Not supported
Data import from Message Queue for Apache Kafka	Supported	Not supported
Data import from self-managed databases, such as MySQL and PostgreSQL databases	Supported	Not supported
Data import from MaxCompute	Supported	Supported

Scenario	Elastic storage mode	Serverless mode
Data import from Object Storage Service (OSS) in a variety of formats such as JSON, Avro, and CSV	Supported	Supported
Data import by using the JDBC or ODBC client	Supported	Supported

The following table describes the data analysis scenarios.

Scenario	Elastic storage mode	Serverless mode
Standard SQL capability	Supported	Supported
Spatio-temporal data analysis based on PostGIS or Ganos	Supported	Not supported
Machine learning	Supported	Supported
Data lake analysis based on OSS or MaxCompute foreign tables	Supported	Supported
Federated analysis based on OSS or MaxCompute foreign tables	Supported	Supported
Batch processing of large amounts of offline data	Supported	Supported
Vector analysis	Supported	Not supported

The following table describes the workload management scenarios.

Scenario	Elastic storage mode	Serverless mode
Resource isolation	Supported	Supported
Cross-tenant database or warehouse isolation	Supported	Supported
Join analysis for multiple instances	Not supported	Supported
Time-specific scaling	Not supported	Not supported

Cases for selecting instance specifications

Case 1: Internet and manufacturing users

Internet and manufacturing users want to migrate data from self-managed databases and Greenplum data warehouses to the cloud.

Suggestion: We recommend that you select in elastic storage mode.

Benefits: is compatible with Greenplum, PostgreSQL, and other open source ecosystems. Data can be seamlessly migrated to AnalyticDB for PostgreSQL. After migration, resources can be adjusted based on business requirements.

Case 2: Internet SaaS users

Internet SaaS users need to build a data mid-end that runs stably. On this data mid-end, they want to perform extract, transform, and load (ETL) operations on a variety of data sources such as ApsaraDB RDS, Realtime Compute for Apache Flink, and OSS, implement hybrid transactional and analytical processing (HTAP), and work with BI reports and enterprise-class data services.

Suggestion: We recommend that you select in elastic storage mode. Recommended instance specifications: High-availability Edition, compute node specifications higher than 4 cores and 32 GB, and more than four compute nodes.

Benefits: The elastic storage mode allows data import from other Alibaba Cloud services or third-party cloud services. It provides enterprise-class capabilities, such as workload management based on user-defined functions or resource queues. The elastic storage mode supports ETL operations and provides computing performance about three times that of traditional data warehouses. It supports changes to applications that supply the data for analysis. It supports node specification changes and storage scaling based on business requirements.

Case 3: digital transformation of traditional enterprises

Traditional enterprises need to perform digital transformation and replace traditional data warehouses such as Teradata, Oracle, Db2, and Greenplum data warehouses on IDCs with cloud services.

Suggestion: We recommend that you select in elastic storage mode. Recommended instance specifications: High-availability Edition or Basic Edition (based on your business requirements), compute node specifications higher than 4 cores and 32 GB, and more than four compute nodes.

Benefits: is an industry-leading service that can replace Teradata and Oracle data warehouses. It has provided successful solutions for hundreds of financial institutions, ISPs, public service sectors, and enterprises.

Case 4: autonomous driving enterprises

Autonomous driving enterprises need to perform geographical and time series analysis on the vehicle-collected data. They require JSON compatibility and spatio-temporal data analysis capabilities to build business dashboards and support feature engineering.

Suggestion: We recommend that you select in elastic storage mode. Recommended instance specifications: Basic Edition and compute node specifications higher than 4 cores and 32 GB.

Benefits: The elastic storage mode supports the PostGIS and Ganos engines for spatio-temporal analysis and can implement accelerated queries in the massively parallel processing (MPP) architecture. It supports flexible analysis on semi-structured data (such as JSON data) and data lake analysis.

Case 5: Internet gaming enterprises

Internet gaming enterprises need to build a data mid-end to analyze user behavior data. The data mid-end must provide business log cleansing and data join analysis to support gaming operations tools. Internet gaming enterprises need to process HTAP workloads and isolate resources during working hours.

Suggestion: We recommend that you select in Serverless mode. Recommended instance specifications: compute node specifications higher than 4 cores and 16 GB and more than four compute nodes.

Benefits: The Serverless mode can adjust resources flexibly to meet business requirements during different time periods. It provides an efficient Log Service + OSS solution for log data cleansing. The Serverless mode has powerful analysis and single-node computing capabilities.

Case 6: new retail enterprises

New retail enterprises need to build a customer data platform (CDP) that is capable of importing data from multiple sources and selecting appropriate customers.

Suggestion: We recommend that you select in elastic storage mode. Recommended instance specifications: High-availability Edition or Basic Edition (based on your business requirements), compute node specifications higher than 4 cores and 32 GB, and more than four compute nodes.

Benefits: The elastic storage mode supports a variety of data formats such as JSON, CSV, Avro, and Parquet to aggregate data and generate tags. It helps implement one-stop cloud-based platform building by working with other in-house services of Alibaba Cloud such as Quick Audience.

Case 7: large-scale Internet enterprises

Large-scale Internet enterprises have independent business mid-ends for each business unit and a unified data mid-end. They hope that independent resources can be efficiently deployed to support different business workloads and data silos are not generated in the future.

Suggestion: We recommend that you select in Serverless mode. Recommended instance specifications: compute node specifications higher than 4 cores and 16 GB and more than two compute nodes. You can deploy multiple instances.

Benefits: The Serverless mode provides efficient resource deployment and elasticity to prevent heavy pre-planning and dynamically deploy workloads. Data can be shared among multiple instances. Because of this, no data silos are generated as a result of the development of business mid-ends and the data system construction. A single instance can completely isolate resources. The resource usage of each business line can be displayed in the bill.

Case 8: building a data development platform

Users need to build a data development platform to reduce the impact of development on business and improve the development efficiency.

Suggestion: We recommend that you select in Serverless mode. Recommended instance specifications: compute node specifications higher than 4 cores and 16 GB and more than two compute nodes. You can deploy multiple instances.

Benefits: in Serverless mode supports the data sharing feature. When you need to perform data development, you can use this feature to consume the data shared by test instances in the production environment. This prevents the impact of development on the production environment and provides up-to-date data for development.

4.Basic Edition

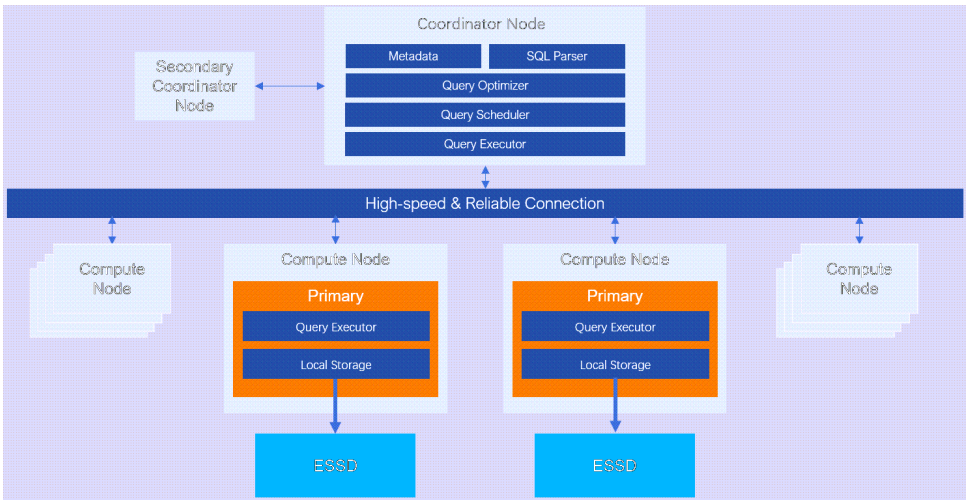
Basic Edition uses a single-replica architecture that reduces costs for storage and entry-level instances and enables high I/O performance.

> **Note** Basic Edition instances can handle most business analytics scenarios. However, for your core business requirements, we recommend that you use High-availability Edition.

Architecture

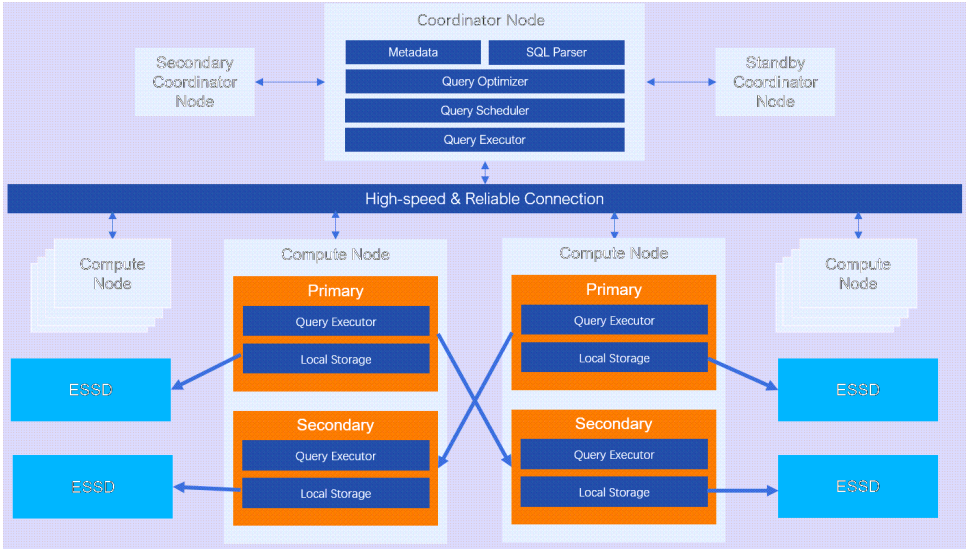
The coordinator nodes and compute nodes of Basic Edition instances are deployed in a single-replica architecture, as shown in the following figure.

Basic Edition architecture



Compared with the High-availability Edition architecture, the Basic Edition architecture does not contain the standby coordinator node or secondary compute nodes.

High-availability Edition architecture



The Basic Edition architecture has the following benefits:

- The storage usage of the standby coordinator node is eliminated.
- The storage usage of compute nodes is reduced by half.
- The data synchronization process between the primary and secondary compute nodes is eliminated, which improves the I/O performance when data is written.

Billing rules

For more information, see AnalyticDB for PostgreSQL Pricing.

Advantages

- **Cost reduction**

  Compared with a High-availability Edition instance, a Basic Edition instance with the same specifications provides the following cost advantages:

  - The storage cost is reduced by 50% because the instance has one less replica.
  - Compute nodes cost less but deliver the same computing power.

Specifications	Storage pricing			Compute node pricing			Instance pricing		
	Basic Edition	High-availability Edition	Reduced by	Basic Edition	High-availability Edition	Reduced by	Basic Edition	High-availability Edition	Reduced by
Entry-level specifications	USD 22.4/month	USD 100/month	77.6%	USD 175.55/month	USD 352.05/month	50.13%	USD 197.95/month	USD 452.05/month	56.21%
Common specifications	USD 89.6/month	USD 200/month	55.2%	USD 668.65/month	USD 700.28/month	4.52%	USD 758.25/month	USD 900.28/month	15.78%

  - A Basic Edition instance with entry-level specifications has 2 CPU cores, 50 GB of storage space, and 2 compute nodes. A High-availability Edition instance with entry-level specifications has 2 CPU cores, 50 GB of storage space, and 4 compute nodes. These are the minimum specifications for AnalyticDB for PostgreSQL instances. For entry-level specifications, the price of a Basic Edition instance is 56% lower than that of a High-availability Edition instance.
  - In common scenarios, an AnalyticDB for PostgreSQL instance has 4 CPU cores, 100 GB of storage space, and 4 compute nodes for both Basic Edition and High-availability Edition. In these scenarios, the price of a Basic Edition instance is 15% lower than that of a High-availability Edition with the same specifications.

- **Performance improvement**

Basic Edition delivers significantly better I/O performance than High-availability Edition. A Basic Edition instance with 2 CPU cores delivers up to 250% higher I/O performance than a High-availability Edition instance with the same specifications. The data synchronization and streaming replication processes are eliminated in Basic Edition. This allows an additional 100% I/O performance improvement in write-intensive scenarios.

In the following examples, the performance advantages of Basic Edition over High-availability Edition are demonstrated in local replication and TPC-H test scenarios. An instance of each edition is used, and both instances have 2 CPU cores, 400 GB of storage space, and 4 compute nodes.

- Local replication

  A row-oriented table with about 90 GB of data is replicated within the instance. The following statement is executed:

  ```
  create table lineitem2 as (select * from lineitem);
  ```

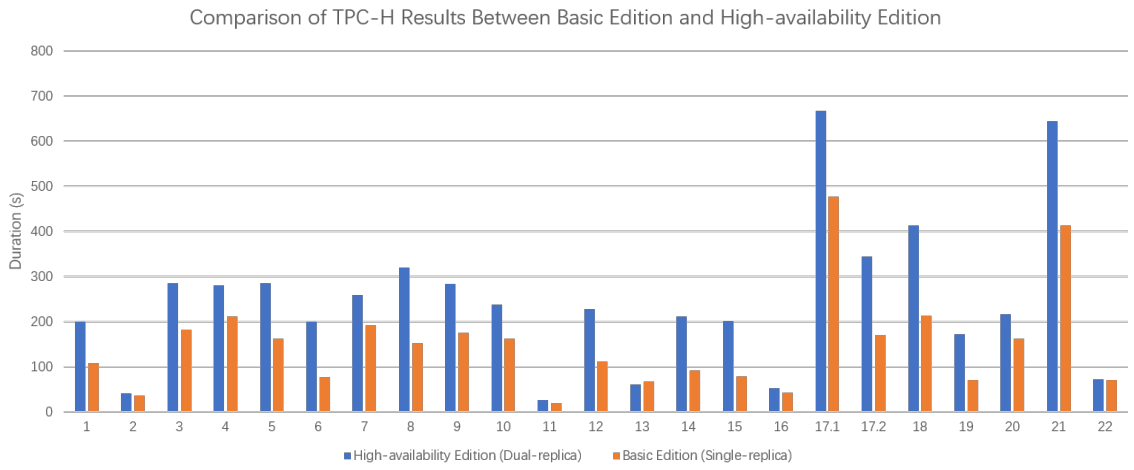
  It takes 249 seconds for the Basic Edition instance to complete the operation but 1,307 seconds for the High-availability Edition instance, which is nearly a five-time performance improvement.

  The test shows that Basic Edition provides significantly better performance in I/O-intensive scenarios, such as CTAS and INSERT INTO SELECT operations.

- TPC-H testing

  > **Note** In this example, a test based on the TPC-H benchmark test is implemented, but it does not meet all the requirements of the TPC-H benchmark test. Therefore, the test results cannot be compared with the published results of the TPC-H benchmark test.

  In this test, 22 SQL statements are executed on a TPC-H test dataset that contains 100 GB of data. The following figure shows the results.



  The time consumed by the Basic Edition instance is 40% less than the High-availability instance, which demonstrates the I/O performance improvement.

Availability

- **Data reliability**

  Enhanced SSDs (ESSDs) are used by to store data. They provide high data reliability even in single-replica mode and ensure data integrity when faults occur on compute nodes.

- **High availability**

  Basic Edition delivers lower availability due to the reduction of a replica. This increases the amount of time required to recover instances in severe scenarios, such as physical machine failures. Basic Edition uses the multi-replica feature of ESSDs to ensure data reliability. The checkpoint mechanism of PostgreSQL is optimized to reduce recovery time for AnalyticDB for PostgreSQL instances.

  The following section compares the availability of Basic Edition and High-availability Edition instances in common failure scenarios:

  - **Failures that trigger the recovery mode**

    In most failure scenarios in , the recovery mode is triggered. The recovery process takes a significantly less amount of time for Basic Edition than for High-availability Edition.

    Most SQL crashes are caused by core dumps or out of memory (OOM) errors. In this case, the instance enters the recovery mode. In recovery mode, the system clears the remaining locks and memory and replays the Write Ahead Log (WAL) files to ensure data integrity. Service provision stops on the instance during the recovery process and resumes after the instance is recovered. A High-availability Edition instance may require 5 to 10 minutes to be recovered, while a Basic Edition instance can be recovered within 10 seconds by using the optimized checkpoint mechanism.

    - WAL

      In , all data changes of a transaction are first recorded in WAL files before the transaction is committed. When a database restores data, WAL files can be replayed to restore the data changes that are committed but not written to the disk.

    - Checkpoint

      A checkpoint marks the point in a transaction before which all data changes have been made on the disk. The database can restore data based on the latest checkpoint. performs checkpoints on a regular basis. When a WAL file reaches a specific length, the system also performs checkpoints.

  - **Compute node failures**

    Basic Edition instances provide lower availability when a compute node fails. When faults occur on a compute node of a High-availability Edition instance, the replica seamlessly takes over to ensure service availability, and the faulty compute node becomes the secondary compute node and restarts in the backend. However, when faults occur on a compute node of a Basic Edition instance, the lack of a replica renders the instance unavailable, and the instance must be restarted for recovery.

  - **Host failures**

    Host failures are severe situations and trigger automatic migration of the host. Even in such scenarios, the replica of a High-availability Edition instance can still take over and ensure the normal running of the instance. The host migration is performed in the backend. However, a Basic Edition instance must be restarted after the host migration is complete. The process may take 15 minutes.

References

- [Notice] AnalyticDB for PostgreSQL Basic Edition Released
- Create an AnalyticDB for PostgreSQL instance

5.Serverless mode

In Serverless mode, provides features such as separation of computing and storage resources, elastic scaling within seconds, and real-time data sharing across instances. These features are implemented based on the resource pooling and massive storage capabilities of cloud infrastructures and traditional parallel processing (MPP), online and offline data processing, and Serverless technologies.

Description

In Serverless mode, decouples computing and storage resources so that they can be scaled with different proportions. Storage resources remain billed on a pay-as-you-go basis, but computing capabilities can be independently scaled to meet business requirements. This reduces storage costs and improves efficiency.

AnalyticDB for PostgreSQL in Serverless mode provides the following advantages over AnalyticDB for PostgreSQL in elastic storage mode:

- Reduces storage costs and enables on-demand resource use. You can use AnalyticDB for PostgreSQL in Serverless mode to implement cost-effective data analysis without the need to migrate your data to other storage media. This mode meets the data analysis requirements of the finance and Internet industries.
- Optimizes high-throughput writes and high-performance batch processing operations with elastic scaling to suit the scenarios in which large amounts of data and large traffic fluctuations are involved.
- Provides the data sharing feature based on the separation of storage and computing resources. Shared data can be accessed from other databases without the need of data export and import. It is easier and more cost-effective than data access in traditional data warehouses.

Precautions

> **Notice** On the International site (alibabacloud.com), AnalyticDB for PostgreSQL instances in Serverless mode can be created only on a pay-as-you-go basis.

is supported in the following regions and zones:

- China:
  - China (Beijing): Zone H and Zone I
  - China (Hangzhou): Zone I and Zone J
  - China (Shanghai): Zone G and Zone F
  - China (Zhangjiakou): Zone C
  - China (Shenzhen): Zone E and Zone F
- Asia Pacific:

  Singapore (Singapore): Zone A and Zone C

Comparison of service modes

The Serverless mode supports most features of the elastic storage mode. The following table describes the differences between these two service modes.

Category	Feature	Elastic storage mode	Serverless mode
Instance management	Basic instance information	Supported	Supported
	Logon to databases by using Data Management (DMS)	Supported	Supported
	Instance creation	Supported	Supported
	Instance release	Supported	Supported
	Instance restart	Supported	Supported
	Instance configuration upgrade or downgrade	Supported	Not supported
	Coordinator node addition or removal	Supported	Supported
	Instance scale-out	Supported	Supported
	Instance scale-in	Not supported	Supported
	Minor version update	Supported	Supported
Account management	Account creation	Supported	Supported
	Password resetting	Supported	Supported
Database connection	Basic connection information	Supported	Supported
	Public endpoint application	Supported	Supported
Monitoring and alerting	Monitoring	Supported	Supported
	Alert rules	Supported	Supported
Data security	Whitelists	Supported	Supported
	SQL audit	Supported	Supported
	SSL encryption	Supported	Supported
	Backup and restoration	Supported	Supported
Configuration	Parameter settings	Supported	Supported

Limits

The Serverless mode supports more than 95% of features of the elastic storage mode. In most cases, the same syntax can be used for both the Serverless and elastic storage modes. Tools such as the JDBC connector, ODBC connector, and psql can be used in Serverless mode in the same way as they are used in elastic storage mode. The following table describes the limits of AnalyticDB for PostgreSQL in Serverless mode on specific features.

Category	Feature	Description
Basic features	ALTER TABLE	• Most features of ALTER TABLE are supported. For example, you can modify the table name, delete column constraints, and add or remove columns. • The data type of columns or the distribution column cannot be modified.
	Indexes	Not supported
	Primary keys	Not supported
	Unique constraints	Not supported
	INSERT ON CONFLICT	Not supported
	Table unlogging	Not supported
	Triggers	Not supported
	Heap tables, append-optimized row-oriented tables, and append-optimized column-oriented tables	Not supported
	Custom data types	Not supported
	Explicit cursors	Supported
Compute engines	ORCA optimizer	Supported
	Laser engine	Supported
Transaction capabilities	Subtransactions	Supported
	Transaction isolation levels	Read Committed and Repeatable Read are supported.
Advanced features	Backup and restoration	Supported
	Materialized views	Not supported
	Auto-vacuum	Supported
	Auto-analyze	Supported
	Elastic scale-out	Supported

Advanced features Category	Feature	Description
	Elastic scale-in	Supported
	GIS/Ganos	Not supported
	Data sharing	Supported

Data migration

You can migrate data from an instance in elastic or reserved storage mode to an instance in Serverless mode. For more information, see Migrate data from an AnalyticDB for PostgreSQL instance in elastic or reserved storage mode to an instance in Serverless mode.

The following table describes the support of the Serverless mode for different data migration types.

Migration type	References	Description
Write data	Use INSERT ON CONFLICT to overwrite data	Not supported
	Use COPY ON CONFLICT to overwrite data	Not supported
	Use the Client SDK	Supported
Migrate table data	Use Data Integration to migrate and batch synchronize data	Supported
	Synchronize data from cloud databases	Not supported
	Synchronize data from self-managed databases	Not supported
	Use a Realtime Compute for Apache Flink cluster to write data to an AnalyticDB for PostgreSQL instance	Not supported You can import data by using external tables.
	Use the \copy command to import data from your computer to AnalyticDB for PostgreSQL	Supported
	Use an external table to import data from OSS at a high speed	Supported
	Use an external table to read and write HDFS data	Supported
	Migrate data from a self-managed Greenplum cluster to an AnalyticDB for PostgreSQL instance	Not supported You can import data by using external tables.

Migration type	References	Description
Migrate warehouse data	Migrate data from a Teradata database to an AnalyticDB for PostgreSQL instance	Not supported You can import data by using external tables.
	Migrate data from an Amazon Redshift instance to an AnalyticDB for PostgreSQL instance	Not supported You can import data by using external tables.
	Migrate data from a self-managed Oracle application to an AnalyticDB for PostgreSQL instance	Not supported You can import data by using external tables.
	Migrate data from a self-managed Oracle database to an AnalyticDB for PostgreSQL instance	Not supported You can import data by using external tables.

Elastic scaling

Instances in Serverless mode can be scaled within minutes. The following scaling performance is provided for reference:

- An instance that has 16 or fewer compute nodes can be scaled within 60 seconds.
- An instance that has more than 16 compute nodes can be scaled within 5 minutes.

You can use the elastic scaling capability of AnalyticDB for PostgreSQL in Serverless mode to scale out compute nodes before expected peak periods such as Double 11 and then scale in compute nodes after the peak hours. is billed on an hourly basis based on the actual duration of resource use and compute node specifications. This way, you can minimize costs while ensuring the service performance.

In Serverless mode, each compute node has a maximum storage capacity. Before you scale in compute nodes, make sure that the total amount of data does not exceed the maximum storage capacity of all the remaining compute nodes combined. For example, assume that each compute node provides 2 CPU cores, 8 GB of memory, and a maximum of 960 GB storage capacity. If you want to scale in your instance to four compute nodes, the total amount of data cannot exceed 3,840 GB (960 GB × 4).

The following table describes the maximum storage capacity of different compute node specifications in AnalyticDB for PostgreSQL in Serverless mode.

Specifications	Maximum storage capacity
2 cores, 8 GB	960 GB
4 cores, 16 GB	2,200 GB
8 cores, 32 GB	5,400 GB
16 cores, 64 GB	11,800 GB

Except during service interruptions that occur before and after scaling, workloads remain readable and writable during the scaling process.

Data sharing (beta)

Compared with the data import and export method used in traditional data warehouses, data sharing used in AnalyticDB for PostgreSQL in Serverless mode has the following advantages:

- Reduced storage costs: Data replication or migration is not needed across instances. Only a single copy of data is stored in the distributed storage and can be shared by multiple instances within the specified range. This way, less storage space is required.
- Ease of use: After you create a share on a producer instance, perform authorization, and then import data to the share, you can access the shared data on a consumer instance in the same manner as you would access data on the producer instance. The table schema does not need to be migrated. The addition or removal of shared objects and authorization changes can be automatically synchronized to the consumer instance.
- Data consistency: Consumer instances can access the shared data with capabilities approximate to those of producer instances. In addition, consumer instances can read the latest written data of producer instances. This ensures the atomicity, consistency, isolation, durability (ACID) capabilities of transactions.

Data sharing can help you resolve the following issues:

- Isolation between complex organization permissions: For example, assume that an instance is created in the corporate headquarters and another instance is created in a branch. Data sharing can be used to allow the instance in the branch to access specific data of the instance in the corporate headquarters.
- Isolation between complex business resources: For example, assume that physical resources are isolated between the extract, transform, load (ETL) and ad hoc business types. Data sharing can be used to share ETL results among instances that involve ad hoc queries.
- Failure on cross-business collaboration: For example, if the same copy of data needs to be analyzed by R&D, sales, operations, and financial personnel, data sharing can be used to allow data access by different business groups within an organization.

Data sharing is in beta testing and has the following limits:

- Data sharing is supported on standard tables. It is not supported on partitioned tables, external tables, views, schemas, or functions.
- Data sharing is supported on hash distributed tables. It is not supported on replicated tables or random tables.
- Data sharing is not supported on subtransactions.
- If multiple shares exist in a producer instance, a consumer instance can subscribe only to one of the shares.
- DDL operations are not allowed on shared tables. To perform DDL operations on a shared table, you must disable sharing for the table.

References

- [Notice] Serverless mode released for AnalyticDB for PostgreSQL
- Overview
- Migrate data from an AnalyticDB for PostgreSQL instance in elastic or reserved storage mode to an instance in Serverless mode

- Performance of the Serverless mode
- Data sharing

6.Performance tests on the Serverless mode

6.1. Performance of the Serverless mode

This topic describes the import and query performance of .

> **Note** The TPC-H performance tests described in this topic are implemented based on the TPC-H benchmark tests but cannot meet all requirements of TPC-H benchmark tests. Therefore, the test results described in this topic are incomparable with the published TPC-H benchmark test results.

Configuration information

An AnalyticDB for PostgreSQL instance in Serverless mode that is used for testing provides the following configurations:

- Compute node specifications: 4 cores, 16 GB
- Number of compute nodes: 4

Data import performance

This test uses two methods to import large tables: COPY and Foreign Data Wrapper (FDW) on Object Storage Service (OSS). The import performance of AnalyticDB for PostgreSQL in Serverless mode is tested by executing different numbers of concurrent queries.

- Test table:

  This test uses the lineitem table for TPC-H and generates 500 GB of test data. For more information about how to generate test data, see TPC-H.

- Test methods:
  - COPY: For more information, see COPY.
  - FDW on OSS: For more information, see Use OSS foreign tables to access OSS data.

The following table describes the test results.

Test method	One concurrent query	Four concurrent queries	Eight concurrent queries
COPY	37 MB/s	125 MB/s	128 MB/s
FDW on OSS	47 MB/s	86 MB/s	110 MB/s

Data query performance

This test uses the QGen tool for TPC-H and respectively generates 10 GB and 500 GB of test data. The query duration of AnalyticDB for PostgreSQL in Serverless mode is tested and compared with that of AnalyticDB for PostgreSQL in elastic storage mode. For more information about the test, see TPC-H.

The following table describes the query performance test results for 10 GB of test data.

Query statement	Elastic storage mode	Serverless mode
Q1	15,215.417 ms	8,468.049 ms
Q2	2,949.254 ms	3,874.710 ms
Q3	3,979.300 ms	2,652.187 ms
Q4	6,059.405 ms	2,561.089 ms
Q5	6,833.062 ms	4,297.496 ms
Q6	482.411 ms	578.026 ms
Q7	6,228.587 ms	4,301.195 ms
Q8	6,544.251 ms	5,011.280 ms
Q9	11,240.953 ms	7,742.912 ms
Q10	3,549.456 ms	2,767.839 ms
Q11	1,361.575 ms	1,488.599 ms
Q12	1,661.359 ms	1,842.725 ms
Q13	5,383.167 ms	5,018.539 ms
Q14	744.585 ms	751.640 ms
Q15	1,344.129 ms	1,897.243 ms
Q16	1,550.342 ms	1,984.808 ms
Q17	19,425.750 ms	15,709.382 ms
Q18	19,417.051 ms	6,803.475 ms
Q19	4,762.443 ms	2,375.202 ms
Q20	3,434.726 ms	3,485.165 ms
Q21	14,496.656 ms	8,104.987 ms
Q22	3,174.644 ms	2,918.874 ms
Total duration	2 minutes and 19.951 seconds	1 minute and 34.748 seconds

The following table describes the query performance test results for 500 GB of test data.

Query statement	Elastic storage mode	Serverless mode
Q1	776,749.919 ms	655,198.377 ms
Q2	127,436.833 ms	87,954.528 ms
Q3	323,528.962 ms	664,481.555 ms
Q4	351,981.303 ms	200,034.509 ms
Q5	427,701.721 ms	609,339.053 ms
Q6	110,562.730 ms	19,149.394 ms
Q7	675,657.163 ms	305,690.833 ms
Q8	516,443.454 ms	1,033,242.151 ms
Q9	1,531,569.731 ms	999,391.734 ms
Q10	295,668.016 ms	141,176.254 ms
Q11	141,573.826 ms	74,402.558 ms
Q12	249,247.709 ms	88,836.774 ms
Q13	315,628.505 ms	177,885.452 ms
Q14	187,791.651 ms	39,034.109 ms
Q15	460,263.848 ms	82,863.306 ms
Q16	123,408.319 ms	54,713.206 ms
Q17	4,650,424.484 ms	2,215,070.817 ms
Q18	1151063.573 ms	548,049.730 ms
Q19	260,702.969 ms	85,419.149 ms
Q20	549,780.389 ms	213,492.958 ms
Q21	1,103,378.860 ms	456,781.416 ms
Q22	223,275.303 ms	86,325.201 ms
Total duration	242 minutes and 34.602 seconds	147 minutes and 19.298 seconds

6.2. Test the performance of DELETE and UPDATE operations for instances in Serverless mode

This topic describes how to test the performance of DELETE and UPDATE operations for AnalyticDB for PostgreSQL instances in Serverless mode.

> **Note** The TPC-H performance tests described in this topic are implemented based on the TPC-H benchmark tests but cannot meet all requirements of TPC-H benchmark tests. Therefore, the test results described in this topic are incomparable with the published TPC-H benchmark test results.

Test instance

For information about how to create an instance, see Create an AnalyticDB for PostgreSQL instance.

Instance specifications of the test instance:

- Serverless mode
  - Compute node specifications: 4 cores, 16 GB
  - Number of compute nodes: 4
- Elastic storage mode
  - Compute node specifications: 2 cores, 16 GB
  - Number of compute nodes: 4

Test data

For information about how to generate and import test data, see TPC-H.

Description of the test data:

- Amount of data: 100 GB
- Test tables: lineitem and orders

  You can execute the following statement to create a table named lineitem:

```
CREATE TABLE lineitem (
    l_orderkey bigint NOT NULL,
    l_partkey integer NOT NULL,
    l_suppkey integer NOT NULL,
    l_linenumber integer NOT NULL,
    l_quantity numeric(15,2) NOT NULL,
    l_extendedprice numeric(15,2) NOT NULL,
    l_discount numeric(15,2) NOT NULL,
    l_tax numeric(15,2) NOT NULL,
    l_returnflag character(1) NOT NULL,
    l_linestatus character(1) NOT NULL,
    l_shipdate date NOT NULL,
    l_commitdate date NOT NULL,
    l_receiptdate date NOT NULL,
    l_shipinstruct character(25) NOT NULL,
    l_shipmode char(10) NOT NULL,
    l_comment varchar(44) NOT NULL
)
distributed by (l_orderkey) order by (l_shipdate,l_orderkey);
```

You can execute the following statement to create a table named orders:

```
CREATE TABLE orders (
    o_orderkey bigint NOT NULL,
    o_custkey integer NOT NULL,
    o_orderstatus character(1) NOT NULL,
    o_totalprice numeric(15,2) NOT NULL,
    o_orderdate date NOT NULL,
    o_orderpriority character(15) NOT NULL,
    o_clerk character(15) NOT NULL,
    o_shippriority integer NOT NULL,
    o_comment character varying(79) NOT NULL
)
distributed by (o_orderkey) order by (o_orderdate, o_orderkey);
```

- Number of rows: 600 million rows in the lineitem table, 150 million rows in the orders table, and totally 750 million rows

  Data in both the lineitem and orders tables is evenly distributed by day. The lineitem table has about 250,000 rows of data per day, and the orders table has about 60,000 rows of data per day. As a result, the time range in the test statements determines the amount of data that is queried. In this test, the following time ranges and amounts of data are used:

  - One day: about 310,000 rows of data
  - Half a month: about 4.66 million rows of data
  - One month: about 9.65 million rows of data

Test statements

DELETE and UPDATE operations are performed based on the time columns of the lineitem and orders tables. Sample statements:

- DELETE statements

- lineitem

```
DELETE FROM lineitem WHERE l_shipdate >= YYYYMMDD AND l_shipdate <= YYYYMMDD;
```

- orders

```
DELETE FROM orders WHERE o_orderdate >= YYYYMMDD AND o_orderdate <= YYYYMMDD;
```

- UPDATE statements
  - lineitem

```
UPDATE lineitem SET l_quantity = 100.00, l_shipmode = 'test', l_comment = 'only_for_test'
WHERE l_shipdate >= YYYYMMDD AND l_shipdate <= YYYYMMDD;
```

  - orders

```
UPDATE orders SET o_totalprice = 100.00, o_shippriority = 10, o_comment = 'only_for_test'
WHERE o_orderdate >= YYYYMMDD AND o_orderdate <= YYYYMMDD;
```

- SELECT statements

  Every time after a DELETE or UPDATE operation is performed, the `SELECT count(1)` statement is executed to test the SELECT performance after data changes.

  - lineitem

```
SELECT count(1) FROM lineitem;
```

  - orders

```
SELECT count(1) FROM orders;
```

> **Note** To use the preceding statements, replace `YYYYMMDD` with the date to query. Example: `'19950505'`.

Test results

Before DELETE and UPDATE operations, the `SELECT count(1)` statement is executed on the lineitem and orders tables. The following table describes the test results.

Table name	Serverless mode	Elastic storage mode
lineitem	40.3s	177.9s
orders	8.0s	40.5s

For each of the instances in Serverless mode and elastic storage mode, DELETE and UPDATE operations are performed on the lineitem and orders tables. Each operation is performed for three times and the average durations are recorded. The following table describes the test results.

Operation	Time range	Total execution duration for the lineitem and orders tables in the instance in Serverless mode	Total execution duration for the lineitem and orders tables in the instance in elastic storage mode	SELECT duration for the instance in Serverless mode	SELECT duration for the instance in elastic storage mode
DELETE	One day	32s	290s	lineitem: 40.7s orders: 8.2s	lineitem: 214.3s orders: 56.8s
	Half a month	38s	827s	lineitem: 41.6s orders: 8.5s	lineitem: 657.1s orders: 170.0s
	One month	40s	994s	lineitem: 41.7s orders: 8.3s	lineitem: 788.9s orders: 197.3s
UPDATE	One day	104s	300s	lineitem: 41.6s orders: 8.5s	lineitem: 206.6s orders: 57.2s
	Half a month	110s	824s	lineitem: 42.7s orders: 8.4s	lineitem: 650.0s orders: 172.4s
	One month	114s	988s	lineitem: 42.9s orders: 8.5s	lineitem: 784.1s orders: 201.8s

Conclusions:

- The execution speed of DELETE and UPDATE operations on the instance in Serverless mode is 3 to 20 times faster than that of the instance in elastic storage mode. The larger the data amount involved, the greater the performance difference.

  Reasons: DELETE and UPDATE operations filter data by using the ORDER BY clause. For tables of the instance in Serverless mode, hybrid row-column storage and ORDER BY result in high filtering efficiency on a single column. For tables of the instance in elastic storage mode, all data must be read when data is filtered by column, which incurs read amplification issues. In addition, binary search is not implemented in column-based filtering. As a result, the execution performance of the instance in elastic storage mode is worse than that of the instance in Serverless mode.
- Compared with the instance in elastic storage mode, the execution speed of the `SELECT count(1)` statement on the instance in Serverless mode is four to five times faster.

Reasons: For tables of the instance in Serverless mode, `SELECT count(1)` operations are performed by column. For tables of the instance in elastic storage mode, the preceding operations must read all data, which incurs read amplification issues.

- The query performance of the `SELECT count(1)` statement is nearly the same on the instance in Serverless mode before and after DELETE and UPDATE operations.

  Reasons: For tables of the instance in Serverless mode, SELECT count(1) operations are performed by column, and data deletion verification is performed against the delete bitmap in a short period of time. No read amplification issues occur.

6.3. Test the performance of data sharing for instances in Serverless mode

This topic describes how to test the performance of data sharing for producer and consumer instances in .

> **Note** The TPC-H performance tests described in this topic are implemented based on the TPC-H benchmark tests but cannot meet all requirements of TPC-H benchmark tests. Therefore, the test results described in this topic are incomparable with the published TPC-H benchmark test results.

Test description

The producer and consumer instances used for data sharing have the same instance specifications.

- Compute node specifications: 4 cores, 16 GB
- Number of compute nodes: 4
- Region and zone: Singapore Zone A
- Minor version: V1.0.1.0

For more information about how to create an instance, see Create an AnalyticDB for PostgreSQL instance.

In this test, the DbGen tool is used to generate 100 GB of raw data. For more information about how to install DbGen and import data, see the "Generate test data" section of the TPC-H topic.

Test procedure

1. Enable data sharing for the producer and consumer instances. For more information, see Enable data sharing for instances.
2. Perform the following operations on the producer instance for preparations:
   i. Connect to the producer instance. For more information, see Use client tools to connect to an instance.

   > **Note** In this test, the psql client is used.

ii. Create a database named db01 and switch to the db01 database.

```
CREATE DATEBASE db01;
\c db01
```

iii. Query the UUID of the db01 database.

```
SELECT current_database_uuid();
```

iv. Create a schema named tpch and set it as the default schema.

```
CREATE SCHEMA IF NOT EXISTS tpch;
SET search_path = tpch;
```

v. Create eight test tables for TPC-H.

```
CREATE TABLE customer (
    c_custkey integer NOT NULL,
    c_name character varying(25) NOT NULL,
    c_address character varying(40) NOT NULL,
    c_nationkey integer NOT NULL,
    c_phone character(15) NOT NULL,
    c_acctbal numeric(15,2) NOT NULL,
    c_mktsegment character(10) NOT NULL,
    c_comment character varying(117) NOT NULL
)
distributed by (c_custkey);
CREATE TABLE lineitem (
    l_orderkey bigint NOT NULL,
    l_partkey integer NOT NULL,
    l_suppkey integer NOT NULL,
    l_linenumber integer NOT NULL,
    l_quantity numeric(15,2) NOT NULL,
    l_extendedprice numeric(15,2) NOT NULL,
    l_discount numeric(15,2) NOT NULL,
    l_tax numeric(15,2) NOT NULL,
    l_returnflag character(1) NOT NULL,
    l_linestatus character(1) NOT NULL,
    l_shipdate date NOT NULL,
    l_commitdate date NOT NULL,
    l_receiptdate date NOT NULL,
    l_shipinstruct character(25) NOT NULL,
    l_shipmode char(10) NOT NULL,
    l_comment varchar(44) NOT NULL
)
distributed by (l_orderkey);
CREATE TABLE nation (
    n_nationkey integer NOT NULL,
    n_name character(25) NOT NULL,
    n_regionkey integer NOT NULL,
    n_comment character varying(152)
)
distributed by (n_nationkey);
CREATE TABLE orders (
    o_orderkey bigint NOT NULL,
    o_custkey integer NOT NULL,
```

```
    o_orderstatus character(1) NOT NULL,
    o_totalprice numeric(15,2) NOT NULL,
    o_orderdate date NOT NULL,
    o_orderpriority character(15) NOT NULL,
    o_clerk character(15) NOT NULL,
    o_shippriority integer NOT NULL,
    o_comment character varying(79) NOT NULL
)
distributed by (o_orderkey);
CREATE TABLE part (
    p_partkey integer NOT NULL,
    p_name character varying(55) NOT NULL,
    p_mfgr character(25) NOT NULL,
    p_brand character(10) NOT NULL,
    p_type character varying(25) NOT NULL,
    p_size integer NOT NULL,
    p_container character(10) NOT NULL,
    p_retailprice numeric(15,2) NOT NULL,
    p_comment character varying(23) NOT NULL
)
distributed by (p_partkey);
CREATE TABLE partsupp (
    ps_partkey integer NOT NULL,
    ps_suppkey integer NOT NULL,
    ps_availqty integer NOT NULL,
    ps_supplycost numeric(15,2) NOT NULL,
    ps_comment character varying(199) NOT NULL
)
distributed by (ps_partkey);
CREATE TABLE region (
    r_regionkey integer NOT NULL,
    r_name character(25) NOT NULL,
    r_comment character varying(152)
)
distributed by (r_regionkey);
CREATE TABLE supplier (
    s_suppkey integer NOT NULL,
    s_name character(25) NOT NULL,
    s_address character varying(40) NOT NULL,
    s_nationkey integer NOT NULL,
    s_phone character(15) NOT NULL,
    s_acctbal numeric(15,2) NOT NULL,
    s_comment character varying(101) NOT NULL
)
distributed by (s_suppkey);
```

vi. Import test data. You can use an Object Storage Service (OSS) external table or the \copy command to import data to . For more information, see Use an external table to import data from OSS at a high speed or Use the \copy command to import data from your computer to AnalyticDB for PostgreSQL.

The following example shows how to run the \copy command to import data. Replace `'/path/to/localfile'` with the actual path that stores your test data.

```
\COPY customer FROM '/path/to/localfile';
\COPY lineitem FROM '/path/to/localfile';
\COPY nation   FROM '/path/to/localfile';
\COPY orders   FROM '/path/to/localfile';
\COPY part     FROM '/path/to/localfile';
\COPY partsupp FROM '/path/to/localfile';
\COPY region   FROM '/path/to/localfile';
\COPY supplier FROM '/path/to/localfile';
```

3. Perform the following operations on the consumer instance for preparations:
   i. Connect to the consumer instance. For more information, see Use client tools to connect to an instance.
   ii. Create a database named db02 and switch to the db02 database.

```
CREATE DATEBASE db02;
\c db02
```

   iii. Query the UUID of the db02 database.

```
SELECT current_database_uuid();
```

4. Perform the following operations on the producer instance for data sharing:
   i. Create a data share.

```
CREATE DATASHARE s01;
```

   ii. Add the eight test tables to the data share.

```
ALTER DATASHARE s01 ADD TABLE tpch_col.supplier;
ALTER DATASHARE s01 ADD TABLE tpch_col.region;
ALTER DATASHARE s01 ADD TABLE tpch_col.partsupp;
ALTER DATASHARE s01 ADD TABLE tpch_col.part;
ALTER DATASHARE s01 ADD TABLE tpch_col.orders;
ALTER DATASHARE s01 ADD TABLE tpch_col.nation;
ALTER DATASHARE s01 ADD TABLE tpch_col.lineitem;
ALTER DATASHARE s01 ADD TABLE tpch_col.customer;
```

   iii. Authorize the db02 database of the consumer instance to consume the data share.

```
GRANT USAGE ON DATASHARE s01 TO DATABASE "db02-uuid";
```

> **Note** Replace `"db02-uuid"` with the UUID of the db02 database obtained in Step 3.

5. Perform the following operations on the consumer instance for data sharing:

i. Import the data share to the consumer instance.

```
IMPORT DATASHARE s01 AS s01a FROM DATABASE "db01-uuid";
```

> **Note** Replace `"db01-uuid"` with the UUID of the db01 database obtained in Step 2.

ii. Analyze the eight test tables contained in the data share.

```
ANALYZE customer;
ANALYZE lineitem;
ANALYZE nation;
ANALYZE orders;
ANALYZE part;
ANALYZE partsupp;
ANALYZE region;
ANALYZE supplier;
```

6. Execute 22 queries of TPC-H.

> **Note** Before the test, set the optimizer parameter to off. For more information about how to modify parameters, see Configure parameters.

```
-- Q1
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date '1998-12-01' - interval '93 day'
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;
-- Q2
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address
```

```
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 23
    and p_type like '%STEEL'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'EUROPE'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
limit 100;
-- Q3
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'MACHINERY'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '1995-03-24'
    and l_shipdate > date '1995-03-24'
group by
```

```
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit 10;
-- Q4
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1996-08-01'
    and o_orderdate < date '1996-08-01' + interval '3' month
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;
-- Q6
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_shipdate >= date '1994-01-01'
    and l_shipdate < date '1994-01-01' + interval '1' year
    and l_discount between 0.06 - 0.01 and 0.06 + 0.01
    and l_quantity < 24;
-- Q7
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(year from l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
```

```
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = 'JORDAN' and n2.n_name = 'INDONESIA')
                or (n1.n_name = 'INDONESIA' and n2.n_name = 'JORDAN')
            )
            and l_shipdate between date '1995-01-01' and date '1996-12-31'
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year;
-- Q8
select
    o_year,
    sum(case
        when nation = 'INDONESIA' then volume
        else 0
    end) / sum(volume) as mkt_share
from
    (
        select
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
            n2.n_name as nation
        from
            part,
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
            region
        where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
```

```
            and r_name = 'ASIA'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01' and date '1996-12-31'
            and p_type = 'STANDARD BRUSHED BRASS'
    ) as all_nations
group by
    o_year
order by
    o_year;
-- Q9
select
    nation,
    o_year,
    sum(amount) as sum_profit
from
    (
        select
            n_name as nation,
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
        from
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
        where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%chartreuse%'
    ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc;
-- Q10
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
```

```
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date '1994-08-01'
    and o_orderdate < date '1994-08-01' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
limit 20;
-- Q11
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    partsupp,
    supplier,
    nation
where
    ps_suppkey = s_suppkey
    and s_nationkey = n_nationkey
    and n_name = 'INDONESIA'
group by
    ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
            select
                sum(ps_supplycost * ps_availqty) * 0.0001000000
            from
                partsupp,
                supplier,
                nation
            where
                ps_suppkey = s_suppkey
                and s_nationkey = n_nationkey
                and n_name = 'INDONESIA'
        )
order by
    value desc;
-- Q12
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
```

```
            or o_orderpriority = '2-HIGH'
            then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority <> '1-URGENT'
            and o_orderpriority <> '2-HIGH'
            then 1
        else 0
    end) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
    and l_shipmode in ('REG AIR', 'TRUCK')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date '1994-01-01'
    and l_receiptdate < date '1994-01-01' + interval '1' year
group by
    l_shipmode
order by
    l_shipmode;
-- Q13
select
    c_count,
    count(*) as custdist
from
    (
        select
            c_custkey,
            count(o_orderkey)
        from
            customer left outer join orders on
                c_custkey = o_custkey
                and o_comment not like '%pending%requests%'
        group by
            c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;
-- Q14
select
    100.00 * sum(case
        when p_type like 'PROMO%'
            then l_extendedprice * (1 - l_discount)
        else 0
    end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
    lineitem,
    part
```

```
where
    l_partkey = p_partkey
    and l_shipdate >= date '1994-11-01'
    and l_shipdate < date '1994-11-01' + interval '1' month;
-- Q15
create view revenue0 (supplier_no, total_revenue) as
    select
        l_suppkey,
        sum(l_extendedprice * (1 - l_discount))
    from
        lineitem
    where
        l_shipdate >= date '1997-10-01'
        and l_shipdate < date '1997-10-01' + interval '3' month
    group by
        l_suppkey;
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    supplier,
    revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            revenue0
    )
order by
    s_suppkey;
drop view revenue0;
-- Q16
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    part
where
    p_partkey = ps_partkey
    and p_brand <> 'Brand#44'
    and p_type not like 'SMALL BURNISHED%'
    and p_size in (36, 27, 34, 45, 11, 6, 25, 16)
    and ps_suppkey not in (
        select
            s_suppkey
        from
```

```
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;
-- Q17
select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#42'
    and p_container = 'JUMBO PACK'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
        where
            l_partkey = p_partkey
    );
-- Q18
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
from
    customer,
    orders,
    lineitem
where
    o_orderkey in (
        select
            l_orderkey
        from
            lineitem
        group by
            l_orderkey having
                sum(l_quantity) > 312
    )
    and c_custkey = o_custkey
```

```
    and o_orderkey = l_orderkey
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate
limit 100;
-- Q19
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    lineitem,
    part
where
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#43'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 5 and l_quantity <= 5 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#45'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 12 and l_quantity <= 12 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#11'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 24 and l_quantity <= 24 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    );
-- Q20
select
    s_name,
    s_address
from
    supplier,
    nation
```

```
where
    s_suppkey in (
        select
            ps_suppkey
        from
            partsupp
        where
            ps_partkey in (
                select
                    p_partkey
                from
                    part
                where
                    p_name like 'magenta%'
            )
            and ps_availqty > (
                select
                    0.5 * sum(l_quantity)
                from
                    lineitem
                where
                    l_partkey = ps_partkey
                    and l_suppkey = ps_suppkey
                    and l_shipdate >= date '1996-01-01'
                    and l_shipdate < date '1996-01-01' + interval '1' year
            )
    )
    and s_nationkey = n_nationkey
    and n_name = 'RUSSIA'
order by
    s_name;
-- Q21
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
```

```
    and not exists (
        select
            *
        from
            lineitem l3
        where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey <> l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = 'MOZAMBIQUE'
group by
    s_name
order by
    numwait desc,
    s_name
limit 100;
-- Q22
select
        cntrycode,
        count(*) as numcust,
        sum(c_acctbal) as totacctbal
from
        (
                select
                        substring(c_phone from 1 for 2) as cntrycode,
                        c_acctbal
                from
                        customer
                where
                        substring(c_phone from 1 for 2) in
                                ('13', '31', '23', '29', '30', '18', '17')
                        and c_acctbal > (
                                select
                                        avg(c_acctbal)
                                from
                                        customer
                                where
                                        c_acctbal > 0.00
                                        and substring(c_phone from 1 for 2) in
                                                ('13', '31', '23', '29', '30', '18', '1
7')
                        )
                        and not exists (
                                select
                                        *
                                from
                                        orders
                                where
                                        o_custkey = c_custkey
                        )
        ) as custsale
group by
        cntrycode
```

```
order by
        cntrycode;
```

Test results

Query	Query duration on the producer instance (unit: seconds)	Query duration on the consumer instance (unit: seconds)
Q1	287.04	291.46
Q2	18.49	20.14
Q3	143.08	169.46
Q4	61.54	72.78
Q5	105.46	152.77
Q6	23.78	32.56
Q7	84.42	96.63
Q8	77.01	87.33
Q9	329.42	340.8
Q10	81.72	89.85
Q11	18.18	18.24
Q12	62.93	70.79
Q13	141.13	146.47
Q14	29.35	38.33
Q15	56.76	74.08
Q16	20.47	20.27
Q17	944.35	960.16
Q18	228.83	256.7
Q19	57.03	65.63
Q20	192.67	199.1
Q21	289.82	303.06
Q22	48.86	57.07
Total duration	3,302.32	3,563.69

Conclusions

In the TPC-H test scenario that involves 100 GB of shared data, the consumer instance can provide more than 90% the query performance of the producer instance.

7.Limits

To ensure the stability and security of instances, is subject to the following limits.

You do not need to perform basic database O&M operations such as ensuring high availability and installing security patches for purchased AnalyticDB for PostgreSQL instances. However, you must take note of the following items:

- Instance upgrade: Instances are read-only while they are being upgraded. A service interruption of up to 30 seconds may occur after the upgrade is complete. Make sure that your application is configured with automatic reconnection policies such as a connection pool.
- Failover: The coordinator and compute nodes each adopts a primary/secondary high-availability architecture. When an error occurs on the primary node, the service is automatically switched over to the secondary node within 30 seconds. A service interruption of about 30 seconds may occur during the failover. Make sure that your application is configured with automatic reconnection policies such as a connection pool.

Metric	Limits

Maximum number of user connections	The maximum number of connections varies with the instance specifications:
	• High-availability Edition:
	  ◦ 2 cores, 16 GB: 600
	  ◦ 4 cores, 32 GB: 800
	  ◦ 8 cores, 64 GB: 1,000
	• Basic Edition:
	  ◦ 2 cores, 8 GB: 250
	  ◦ 4 cores, 16 GB: 350
	  ◦ 8 cores, 32 GB: 450
	  ◦ 16 cores, 64 GB: 950

Maximum data volume in a column	1 GB
Maximum data volume of a database	The maximum data volume of a database varies with the instance specifications.
Maximum data volume of a table	128 TB per partition
Maximum volume of BLOB data	1 GB **Note** In , the BYTEA data type is used in place of BLOB.
Maximum number of rows in a table	2^48
Maximum number of columns in a table	1,600

Metric	Limits
Maximum number of tables in a database	4.2 billion
Maximum number of columns in a view	1,664
Maximum number of characters in the name of a row, table, or database	128
Maximum number of characters in the name of an object (database, user, base table, view, index, stored procedure, UDF, UDT, constraint, or column)	63
Maximum number of columns in a primary or secondary index	32
Maximum number of columns specified in a SELECT statement	1,664
Trigger	Not supported

8.Terms

The following table lists the basic terms of .

Term	Description
Massively Parallel Processing (MPP)	A distributed shared-nothing computing architecture. MPP uses a large number of nodes that do not share resources with each other to perform parallel computing and improve performance. An AnalyticDB for PostgreSQL instance is composed of multiple compute nodes in the MPP architecture.
compute node	The unit for allocating resources in AnalyticDB for PostgreSQL. An instance is composed of multiple compute nodes. When the number of compute nodes increases, the storage capacity scales out but the query response time does not change. A compute node is a unit of computing resources that includes fixed CPU cores, memory, and storage. Each compute node contains a data partition in the MPP architecture.
number of compute nodes	The number of compute nodes purchased for an instance. A single instance supports up to 4,096 nodes. The storage capacity and computing resources of instances increase linearly with the number of compute nodes.
data distribution	Table data is distributed in different data partitions by partition key. A data partition is a unit for computing and storing data in the MPP architecture. Common data distribution methods include hash distribution, random distribution, and replication distribution.