ALIBABA CLOUD

Alibaba Cloud

AnalyticDB for PostgreSQL
Beginner Developer Guide

Document Version: 20200818

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.Data types

This topic provides an overview of the data types supported by AnalyticDB for PostgreSQL. You can also create a data type by executing a CREATE TYPE statement.

Data types supported

The following table lists the data types supported by AnalyticDB for PostgreSQL.

Data type	Alias	Length	Range	Description
bigint	int8	8 bytes	-9223372036854775808 to 9223372036854775807	An integer within a large range.
bigserial	serial8	8 bytes	1 to 9223372036854775807	A large auto-increment integer.
bit [(n)]		n bits	Bit string constant	A bit string with a fixed length.
bit varying [(n)]	varbit	Variable	Bit string constant	A bit string with a variable length.
boolean	bool	1 byte	true/false, t/f, yes/no, y/n, 1/0	A boolean value (true or false).
box		32 bytes	((x1,y1),(x2,y2))	A rectangular box on a plane, not allowed in a column that is used as the distribution key.
bytea		1 byte + binary string	Sequence of octets	A binary string with a variable length.
character [(n)]	char [(n)]	1 byte + n	String up to n characters in length	A blank-padded string with a fixed length.

Data type	Alias	Length	Range	Description
character varying [(n)]	varchar [(n)]	1 byte + string size	String up to n characters in length	A string with a limited variable length.
cidr		12 or 24 bytes		IPv4 and IPv6 networks.
circle		24 bytes	<(x,y),r> (center and radius)	A circle on a plane, not allowed in a column that is used as the distribution key.
date		4 bytes	4713 BC to 294,277 AD	A calendar date (year, month, day).
decimal [(p, s)]	numeric [(p, s)]	Variable	Unlimited	User-specified precision, which is accurate.
double precision	float8	8 bytes	15 digits	Variable precision, which is inaccurate.
	float			
inet		12 or 24 bytes		IPv4 and IPv6 hosts and networks.
Integer	int or int4	4 bytes	-2.1E+09 to +2147483647	An integer in typical cases.
interval [(p)]		12 bytes	-178000000 years to 178000000 years	A time range.
json		1 byte + JSON size	JSON string	A string with an unlimited variable length.
lseg		32 bytes	((x1,y1),(x2,y2))	A line segment on a plane, not allowed in a column that is used as the distribution key.

Data type	Alias	Length	Range	Description
macaddr		6 bytes		A MAC address.
money		8 bytes	-92233720368547758.08 to +92233720368547758.07	An amount of money.
path		16+16n bytes	[(x1,y1),...]	A geometric path on a plane, not allowed in a column that is used as the distribution key.
point		16 bytes	(x,y)	A geometric point on a plane, not allowed in a column that is used as the distribution key.
polygon		40+16n bytes	((x1,y1),...)	A closed geometric path on a plane, not allowed in a column that is used as the distribution key.
real	float4	4 bytes	6 digits	Variable precision, which is inaccurate.
serial	serial4	4 bytes	1 to 2147483647	An auto-increment integer.
smallint	int2	2 bytes	-32768 to +32767	An integer within a small range.

Data type	Alias	Length	Range	Description
text		1 byte + string size	Unlimited	A string with an unlimited variable length.
time [(p)] [without time zone]		8 bytes	00:00:00[.000000] to 24:00:00[.000000]	The time of a day without the time zone.
time [(p)] with time zone	timetz	12 bytes	00:00:00+1359 to 24:00:00-1359	The time of a day with the time zone.
timestamp [(p)] [without time zone]		8 bytes	4713 BC to 294,277 AD	The date and time without the time zone.
timestamp [(p)] with time zone	timestamptz	8 bytes	4713 BC to 294,277 AD	The date and time with the time zone.
xml		1 byte + XML size	Unlimited	A string with an unlimited variable length.
uuid		32 bytes		The uuid data type is provided with AnalyticDB for PostgreSQL V6.0. In AnalyticDB for PostgreSQL V4.3, however, you must install the uuid-ossp extension before you can use the uuid data type. For more information, see Use the uuid-ossp extension.

References

For more information, visit Pivotal Greenplum documentation.

2.SQL reference

This topic describes the SQL statements that are available in AnalyticDB for PostgreSQL and the syntax used.

- CREATE OPERATOR
- CREATE RESOURCE QUEUE
- CREATE ROLE
- CREATE RULE
- CREATE SCHEMA
- CREATE SEQUENCE
- CREATE TABLE
- CREATE TABLE AS
- CREATE TYPE
- CREATE USER
- CREATE VIEW
- DEALLOCATE
- DECLARE
- DELETE
- DROP AGGREGATE
- DROP CAST
- DROP CONVERSION
- DROP DATABASE
- DROP DOMAIN
- DROP EXTENSION
- DROP EXTERNAL TABLE
- DROP FUNCTION
- DROP GROUP
- DROP INDEX
- DROP LIBRARY
- DROP OPERATOR
- DROP OWNED
- DROP RESOURCE QUEUE
- DROP ROLE
- DROP RULE
- DROP SCHEMA
- DROP SEQUENCE
- DROP TABLE
- DROP TYPE
- DROP USER
- DROP VIEW
- END
- EXECUTE
- EXPLAIN
- FETCH

- GRANT
- INSERT
- LOAD
- LOCK
- MOVE
- PREPARE
- REASSIGN OWNED
- REINDEX
- RELEASE SAVEPOINT
- RESET
- REVOKE
- ROLLBACK
- ROLLBACK TO SAVEPOINT
- SAVEPOINT
- SELECT
- SELECT INTO
- SET
- SET ROLE
- SET SESSION AUTHORIZATION
- SET TRANSACTION
- SHOW
- START TRANSACTION
- TRUNCATE
- UPDATE
- VACUUM
- VALUES

ABORT

Aborts the current transaction.

```
ABORT [WORK | TRANSACTION]
```

For more information, visit ABORT.

ALTER AGGREGATE

Changes the definition of an aggregate function.

```
ALTER AGGREGATE name ( type [ , ... ] ) RENAME TO new_name
ALTER AGGREGATE name ( type [ , ... ] ) OWNER TO new_owner
ALTER AGGREGATE name ( type [ , ... ] ) SET SCHEMA new_schema
```

For more information, visit ALTER AGGREGATE.

ALTER CONVERSION

Changes the definition of a conversion.

```
ALTER CONVERSION name RENAME TO newname
ALTER CONVERSION name OWNER TO newowner
```

For more information, visit ALTER CONVERSION.

ALTER DATABASE

Changes the attributes of a database.

```
ALTER DATABASE name [ WITH CONNECTION LIMIT connlimit ]
ALTER DATABASE name SET parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name RESET parameter
ALTER DATABASE name RENAME TO newname
ALTER DATABASE name OWNER TO new_owner
```

For more information, visit ALTER DATABASE.

ALTER DOMAIN

Changes the definition of a domain.

```
ALTER DOMAIN name { SET DEFAULT expression | DROP DEFAULT }
ALTER DOMAIN name { SET | DROP } NOT NULL
ALTER DOMAIN name ADD domain_constraint
ALTER DOMAIN name DROP CONSTRAINT constraint_name [RESTRICT | CASCADE]
ALTER DOMAIN name OWNER TO new_owner
ALTER DOMAIN name SET SCHEMA new_schema
```

For more information, visit ALTER DOMAIN.

ALTER EXTERNAL TABLE

Changes the definition of an external table.

```
ALTER EXTERNAL TABLE name RENAME [COLUMN] column TO new_column
ALTER EXTERNAL TABLE name RENAME TO new_name
ALTER EXTERNAL TABLE name SET SCHEMA new_schema
ALTER EXTERNAL TABLE name action [, ... ]
```

For more information, visit ALTER EXTERNAL TABLE.

ALTER FUNCTION

Changes the definition of a function.

```
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
action [, ... ] [RESTRICT]
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
RENAME TO new_name
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
OWNER TO new_owner
ALTER FUNCTION name ( [ [argmode] [argname] argtype [, ...] ] )
SET SCHEMA new_schema
```

For more information, visit ALTER FUNCTION.

ALTER GROUP

Changes a role name or membership.

```
ALTER GROUP groupname ADD USER username [, ... ]
ALTER GROUP groupname DROP USER username [, ... ]
ALTER GROUP groupname RENAME TO newname
```

For more information, visit ALTER GROUP.

ALTER INDEX

Changes the definition of an index.

```
ALTER INDEX name RENAME TO new_name
ALTER INDEX name SET TABLESPACE tablespace_name
ALTER INDEX name SET ( FILLFACTOR = value )
ALTER INDEX name RESET ( FILLFACTOR )
```

For more information, visit ALTER INDEX.

ALTER OPERATOR

Changes the definition of an operator.

```
ALTER OPERATOR name ( {lefttype | NONE} , {righttype | NONE} )
OWNER TO newowner
```

For more information, visit ALTER OPERATOR.

ALTER RESOURCE QUEUE

Changes the limits of a resource queue.

```
ALTER RESOURCE QUEUE name WITH ( queue_attribute=value [, ... ] )
```

For more information, visit ALTER RESOURCE QUEUE.

ALTER ROLE

Changes the definition of a database role.

```
ALTER ROLE name RENAME TO newname

ALTER ROLE name SET config_parameter {TO | =} {value | DEFAULT}

ALTER ROLE name RESET config_parameter

ALTER ROLE name RESOURCE QUEUE {queue_name | NONE}

ALTER ROLE name [ [WITH] option [ ... ] ]
```

For more information, visit ALTER ROLE.

ALTER SCHEMA

Changes the definition of a schema.

```
ALTER SCHEMA name RENAME TO newname

ALTER SCHEMA name OWNER TO newowner
```

For more information, visit ALTER SCHEMA.

ALTER SEQUENCE

Changes the definition of a sequence generator.

```
ALTER SEQUENCE name [INCREMENT [ BY ] increment]
[MINVALUE minvalue | NO MINVALUE]
[MAXVALUE maxvalue | NO MAXVALUE]
[RESTART [ WITH ] start]
[CACHE cache] [[ NO ] CYCLE]
[OWNED BY {table.column | NONE}]
ALTER SEQUENCE name SET SCHEMA new_schema
```

For more information, visit ALTER SEQUENCE.

ALTER TABLE

Changes the definition of a table.

```
ALTER TABLE [ONLY] name RENAME [COLUMN] column TO new_column

ALTER TABLE name RENAME TO new_name

ALTER TABLE name SET SCHEMA new_schema

ALTER TABLE [ONLY] name SET
DISTRIBUTED BY (column, [ ... ] )
| DISTRIBUTED RANDOMLY
| WITH (REORGANIZE=true|false)

ALTER TABLE [ONLY] name action [, ... ]

ALTER TABLE name
[ ALTER PARTITION { partition_name | FOR (RANK(number))
| FOR (value) } partition_action [...] ]
partition_action
```

For more information, visit ALTER TABLE.

ALTER TYPE

Changes the definition of a data type.

```
ALTER TYPE name
OWNER TO new_owner | SET SCHEMA new_schema
```

For more information, visit ALTER TYPE.

ALTER USER

Changes the definition of a database role (user).

```
ALTER USER name RENAME TO newname

ALTER USER name SET config_parameter {TO | =} {value | DEFAULT}

ALTER USER name RESET config_parameter

ALTER USER name [ [WITH] option [ ... ] ]
```

For more information, visit ALTER USER.

ANALYZE

Collects statistics about a database.

```
ANALYZE [VERBOSE] [ROOTPARTITION [ALL] ]
[table [ (column [, ...] ) ]]
```

For more information, visit ANALYZE.

BEGIN

Starts a transaction block.

```
BEGIN [WORK | TRANSACTION] [transaction_mode]
[READ ONLY | READ WRITE]
```

For more information, visit BEGIN.

CHECKPOINT

Forces a transaction log checkpoint.

```
CHECKPOINT
```

For more information, visit CHECKPOINT.

CLOSE

Closes a cursor.

```
CLOSE cursor_name
```

For more information, visit CLOSE.

CLUSTER

Physically reorders heap storage tables on a disk based on an index. We recommend that you do not use this statement.

```
CLUSTER indexname ON tablename

CLUSTER tablename

CLUSTER
```

For more information, visit CLUSTER.

COMMENT

Defines or changes the comment of an object.

```
COMMENT ON
{ TABLE object_name |
COLUMN table_name.column_name |
AGGREGATE agg_name (agg_type [, ...]) |
CAST (sourcetype AS targettype) |
CONSTRAINT constraint_name ON table_name |
CONVERSION object_name |
DATABASE object_name |
DOMAIN object_name |
FILESPACE object_name |
FUNCTION func_name ([[argmode] [argname] argtype [, ...]]) |
INDEX object_name |
LARGE OBJECT large_object_oid |
OPERATOR op (leftoperand_type, rightoperand_type) |
OPERATOR CLASS object_name USING index_method |
[PROCEDURAL] LANGUAGE object_name |
RESOURCE QUEUE object_name |
ROLE object_name |
RULE rule_name ON table_name |
SCHEMA object_name |
SEQUENCE object_name |
TABLESPACE object_name |
TRIGGER trigger_name ON table_name |
TYPE object_name |
VIEW object_name }
IS 'text'
```

For more information, visit COMMENT.

COMMIT

Commits the current transaction.

```
COMMIT [WORK | TRANSACTION]
```

For more information, visit COMMIT.

COPY

Copies data between a file and a table.

```
COPY table [(column [, ...])] FROM {'file' | STDIN}
[ [WITH]
[BINARY]
[OIDS]
[HEADER]
[DELIMITER [ AS ] 'delimiter']
[NULL [ AS ] 'null string']
[ESCAPE [ AS ] 'escape' | 'OFF']
[NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
[CSV [QUOTE [ AS ] 'quote']
[FORCE NOT NULL column [, ...]]
[FILL MISSING FIELDS]
[[LOG ERRORS]
SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]

COPY {table [(column [, ...])] | (query)} TO {'file' | STDOUT}
[ [WITH]
[ON SEGMENT]
[BINARY]
[OIDS]
[HEADER]
[DELIMITER [ AS ] 'delimiter']
[NULL [ AS ] 'null string']
[ESCAPE [ AS ] 'escape' | 'OFF']
[CSV [QUOTE [ AS ] 'quote']
[FORCE QUOTE column [, ...]] ]
[IGNORE EXTERNAL PARTITIONS ]
```

For more information, visit COPY.

CREATE AGGREGATE

Creates an aggregate function.

```
CREATE [ORDERED] AGGREGATE name (input_data_type [ , ... ])
( SFUNC = sfunc,
STYPE = state_data_type
[, PREFUNC = prefunc]
[, FINALFUNC = ffunc]
[, INITCOND = initial_condition]
[, SORTOP = sort_operator] )
```

For more information, visit CREATE AGGREGATE.

CREATE CAST

Creates a cast.

```
CREATE CAST (sourcetype AS targettype)
WITH FUNCTION funcname (argtypes)
[AS ASSIGNMENT | AS IMPLICIT]

CREATE CAST (sourcetype AS targettype) WITHOUT FUNCTION
[AS ASSIGNMENT | AS IMPLICIT]
```

For more information, visit CREATE CAST.

CREATE CONVERSION

Creates an encoding conversion.

```
CREATE [DEFAULT] CONVERSION name FOR source_encoding TO
dest_encoding FROM funcname
```

For more information, visit CREATE CONVERSION.

CREATE DATABASE

Creates a database.

```
CREATE DATABASE name [ [WITH] [OWNER [=] dbowner]
[TEMPLATE [=] template]
[ENCODING [=] encoding]
[CONNECTION LIMIT [=] connlimit ] ]
```

For more information, visit CREATE DATABASE.

CREATE DOMAIN

Creates a domain.

```
CREATE DOMAIN name [AS] data_type [DEFAULT expression]
[CONSTRAINT constraint_name
| NOT NULL | NULL
| CHECK (expression) [...]]
```

For more information, visit CREATE DOMAIN.

CREATE EXTENSION

Creates an extension in a database.

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
[ VERSION version ]
[ FROM old_version ]
[ CASCADE ]
```

For more information, visit CREATE EXTENSION.

CREATE EXTERNAL TABLE

Creates an external table.

```
CREATE [READABLE] EXTERNAL TABLE tablename
( columnname datatype [, ...] | LIKE othertable )
LOCATION ('ossprotocol')
FORMAT 'TEXT'
[( [HEADER]
[DELIMITER [AS] 'delimiter' | 'OFF']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF']
[NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
[FILL MISSING FIELDS] )]
| 'CSV'
[( [HEADER]
[QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE NOT NULL column [, ...]]
[ESCAPE [AS] 'escape']
[NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
[FILL MISSING FIELDS] )]
[ ENCODING 'encoding' ]
[ [LOG ERRORS [INTO error_table]] SEGMENT REJECT LIMIT count
[ROWS | PERCENT] ]
CREATE WRITABLE EXTERNAL TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION ('ossprotocol')
FORMAT 'TEXT'
[( [DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[ESCAPE [AS] 'escape' | 'OFF'] )]
```

```
| 'CSV'
[([QUOTE [AS] 'quote']
[DELIMITER [AS] 'delimiter']
[NULL [AS] 'null string']
[FORCE QUOTE column [, ...]] ]
[ESCAPE [AS] 'escape'] )]
[ ENCODING 'encoding' ]
[ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
ossprotocol:
oss://oss_endpoint prefix=prefix_name
id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|false]
ossprotocol:
oss://oss_endpoint dir=[folder/[folder/]...]/file_name
id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|false]
ossprotocol:
oss://oss_endpoint filepath=[folder/[folder/]...]/file_name
id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|false]
```

For more information, visit CREATE EXTERNAL TABLE.

CREATE FUNCTION

Creates a function.

```
CREATE [OR REPLACE] FUNCTION name
( [ [argmode] [argname] argtype [ { DEFAULT | = } defexpr ] [, ...] ] )
[ RETURNS { [ SETOF ] rettype
| TABLE ([{ argname argtype | LIKE other table }
[, ...]])
} ]
{ LANGUAGE langname
| IMMUTABLE | STABLE | VOLATILE
| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINE
| COST execution_cost
| SET configuration_parameter { TO value | = value | FROM CURRENT }
| AS 'definition'
| AS 'obj_file', 'link_symbol' } ...
[ WITH ({ DESCRIBE = describe_function
} [, ...] ) ]
```

For more information, visit CREATE FUNCTION.

CREATE GROUP

Creates a database role.

```
CREATE GROUP name [ [WITH] option [ ... ] ]
```

For more information, visit CREATE GROUP.

CREATE INDEX

Creates an index.

```
CREATE [UNIQUE] INDEX name ON table
[USING btree|bitmap|gist]
( {column | (expression)} [opclass] [, ...] )
[ WITH ( FILLFACTOR = value ) ]
[TABLESPACE tablespace]
[WHERE predicate]
```

For more information, visit CREATE INDEX.

CREATE LIBRARY

Creates a custom software table.

```
CREATE LIBRARY library_name LANGUAGE [JAVA] FROM oss_location OWNER ownername
CREATE LIBRARY library_name LANGUAGE [JAVA] VALUES file_content_hex OWNER ownername
```

For more information, visit CREATE LIBRARY.

CREATE OPERATOR

Creates an operator.

```
CREATE OPERATOR name (
PROCEDURE = funcname
[, LEFTARG = lefttype] [, RIGHTARG = righttype]
[, COMMUTATOR = com_op] [, NEGATOR = neg_op]
[, RESTRICT = res_proc] [, JOIN = join_proc]
[, HASHES] [, MERGES]
[, SORT1 = left_sort_op] [, SORT2 = right_sort_op]
[, LTCMP = less_than_op] [, GTCMP = greater_than_op] )
```

For more information, visit CREATE OPERATOR.

CREATE RESOURCE QUEUE

Creates a resource queue.

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
```

For more information, visit CREATE RESOURCE QUEUE.

CREATE ROLE

Creates a database role (user or group).

```
CREATE ROLE name [[WITH] option [ ... ]]
```

For more information, visit CREATE ROLE.

CREATE RULE

Creates a rewrite rule.

```
CREATE [OR REPLACE] RULE name AS ON event
TO table [WHERE condition]
DO [ALSO | INSTEAD] { NOTHING | command | (command; command
...) }
```

For more information, visit CREATE RULE.

CREATE SCHEMA

Creates a schema.

```
CREATE SCHEMA schema_name [AUTHORIZATION username]
[schema_element [ ... ]]

CREATE SCHEMA AUTHORIZATION rolename [schema_element [ ... ]]
```

For more information, visit CREATE SCHEMA.

CREATE SEQUENCE

Creates a sequence generator.

```
CREATE [TEMPORARY | TEMP] SEQUENCE name
[INCREMENT [BY] value]
[MINVALUE minvalue | NO MINVALUE]
[MAXVALUE maxvalue | NO MAXVALUE]
[START [ WITH ] start]
[CACHE cache]
[[NO] CYCLE]
[OWNED BY { table.column | NONE }]
```

For more information, visit CREATE SEQUENCE.

CREATE TABLE

Creates a table.

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ]
[column_constraint [ ... ]
[ ENCODING ( storage_directive [,...] ) ]
]
| table_constraint
| LIKE other_table [{INCLUDING | EXCLUDING}
{DEFAULTS | CONSTRAINTS}] ...}
[, ... ] ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter=value [, ... ] )
[ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
[ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
[ PARTITION BY partition_type (column)
[ SUBPARTITION BY partition_type (column) ]
[ SUBPARTITION TEMPLATE ( template_spec ) ]
[...]
( partition_spec )
| [ SUBPARTITION BY partition_type (column) ]
[...]
( partition_spec
[ ( subpartition_spec
[(...)]
) ]
)
```

For more information, visit CREATE TABLE.

CREATE TABLE AS

Creates a table from the results of a query.

```
CREATE [ [GLOBAL | LOCAL] {TEMPORARY | TEMP} ] TABLE table_name
[(column_name [, ...] )]
[ WITH ( storage_parameter=value [, ... ] ) ]
[ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP}]
[TABLESPACE tablespace]
AS query
[DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY]
```

For more information, visit CREATE TABLE AS.

CREATE TYPE

Creates a data type.

```
CREATE TYPE name AS ( attribute_name data_type [, ... ] )

CREATE TYPE name AS ENUM ( 'label' [, ... ] )

CREATE TYPE name (
INPUT = input_function,
OUTPUT = output_function
[, RECEIVE = receive_function]
[, SEND = send_function]
[, TYPMOD_IN = type_modifier_input_function ]
[, TYPMOD_OUT = type_modifier_output_function ]
[, INTERNALLENGTH = {internallength | VARIABLE}]
[, PASSEDBYVALUE]
[, ALIGNMENT = alignment]
[, STORAGE = storage]
[, DEFAULT = default]
[, ELEMENT = element]
[, DELIMITER = delimiter] )

CREATE TYPE name
```

For more information, visit CREATE TYPE.

CREATE USER

Creates a database role with the LOGIN permission by default.

```
CREATE USER name [ [WITH] option [ ... ] ]
```

For more information, visit CREATE USER.

CREATE VIEW

Creates a view.

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW name
[ ( column_name [, ...] ) ]
AS query
```

For more information, visit CREATE VIEW.

DEALLOCATE

Cancels the allocation of a prepared statement.

```
DEALLOCATE [PREPARE] name
```

For more information, visit DEALLOCATE.

DECLARE

Defines a cursor.

```
DECLARE name [BINARY] [INSENSITIVE] [NO SCROLL] CURSOR
[{WITH | WITHOUT} HOLD]
FOR query [FOR READ ONLY]
```

For more information, visit DECLARE.

DELETE

Deletes rows from a table.

```
DELETE FROM [ONLY] table [[AS] alias]
[USING usinglist]
[WHERE condition | WHERE CURRENT OF cursor_name ]
```

For more information, visit DELETE.

DROP AGGREGATE

Deletes an aggregate function.

```
DROP AGGREGATE [IF EXISTS] name ( type [, ...] ) [CASCADE | RESTRICT]
```

For more information, visit DROP AGGREGATE.

DROP CAST

Deletes a cast.

```
DROP CAST [IF EXISTS] (sourcetype AS targettype) [CASCADE | RESTRICT]
```

For more information, visit DROP CAST.

DROP CONVERSION

Deletes a conversion.

```
DROP CONVERSION [IF EXISTS] name [CASCADE | RESTRICT]
```

For more information, visit DROP CONVERSION.

DROP DATABASE

Deletes a database.

```
DROP DATABASE [IF EXISTS] name
```

For more information, visit DROP DATABASE.

DROP DOMAIN

Deletes a domain.

```
DROP DOMAIN [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP DOMAIN.

DROP EXTENSION

Deletes an extension from a database.

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

For more information, visit DROP_EXTENSION.

DROP EXTERNAL TABLE

Deletes the definition of an external table.

```
DROP EXTERNAL [WEB] TABLE [IF EXISTS] name [CASCADE | RESTRICT]
```

For more information, visit DROP EXTERNAL TABLE.

DROP FUNCTION

Deletes a function.

```
DROP FUNCTION [IF EXISTS] name ( [ [argmode] [argname] argtype
[, ...] ] ) [CASCADE | RESTRICT]
```

For more information, visit DROP FUNCTION.

DROP GROUP

Deletes a database role.

```
DROP GROUP [IF EXISTS] name [, ...]
```

For more information, visit DROP GROUP.

DROP INDEX

Deletes an index.

```
DROP INDEX [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP INDEX.

DROP LIBRARY

Deletes a custom software table.

```
DROP LIBRARY library_name
```

For more information, visit DROP_LIBRARY.

DROP OPERATOR

Deletes an operator.

```
DROP OPERATOR [IF EXISTS] name ( {lefttype | NONE} ,
{righttype | NONE} ) [CASCADE | RESTRICT]
```

For more information, visit DROP OPERATOR.

DROP OWNED

Deletes database objects owned by a database role.

```
DROP OWNED BY name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP OWNED.

DROP RESOURCE QUEUE

Deletes a resource queue.

```
DROP RESOURCE QUEUE queue_name
```

For more information, visit DROP RESOURCE QUEUE.

DROP ROLE

Deletes a database role.

```
DROP ROLE [IF EXISTS] name [, ...]
```

For more information, visit DROP ROLE.

DROP RULE

Deletes a rewrite rule.

```
DROP RULE [IF EXISTS] name ON relation [CASCADE | RESTRICT]
```

For more information, visit DROP RULE.

DROP SCHEMA

Deletes a schema.

```
DROP SCHEMA [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP SCHEMA.

DROP SEQUENCE

Deletes a sequence.

```
DROP SEQUENCE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP SEQUENCE.

DROP TABLE

Deletes a table.

```
DROP TABLE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP TABLE.

DROP TYPE

Deletes a data type.

```
DROP TYPE [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP TYPE.

DROP USER

Deletes a database role.

```
DROP USER [IF EXISTS] name [, ...]
```

For more information, visit DROP USER.

DROP VIEW

Deletes a view.

```
DROP VIEW [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit DROP VIEW.

END

Commits the current transaction.

```
END [WORK | TRANSACTION]
```

For more information, visit END.

EXECUTE

Executes a prepared SQL statement.

```
EXECUTE name [ (parameter [, ...] ) ]
```

For more information, visit EXECUTE.

EXPLAIN

Shows the query plan of a statement.

```
EXPLAIN [ANALYZE] [VERBOSE] statement
```

For more information, visit EXPLAIN.

FETCH

Retrieves rows from a query by using a cursor.

```
FETCH [ forward_direction { FROM | IN } ] cursorname
```

For more information, visit FETCH.

GRANT

Grants permissions for a database role.

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES |
TRIGGER | TRUNCATE } [,...] | ALL [PRIVILEGES] }
ON [TABLE] tablename [, ...]
TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { {USAGE | SELECT | UPDATE} [,...] | ALL [PRIVILEGES] }
ON SEQUENCE sequencename [, ...]
TO { rolename | PUBLIC } [, ...] [WITH GRANT OPTION]

GRANT { {CREATE | CONNECT | TEMPORARY | TEMP} [,...] | ALL
[PRIVILEGES] }
ON DATABASE dbname [, ...]
TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { EXECUTE | ALL [PRIVILEGES] }
ON FUNCTION funcname ( [ [argmode] [argname] argtype [, ...]
] ) [, ...]
TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { USAGE | ALL [PRIVILEGES] }
ON LANGUAGE langname [, ...]
TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { {CREATE | USAGE} [,...] | ALL [PRIVILEGES] }
ON SCHEMA schemaname [, ...]
TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT { CREATE | ALL [PRIVILEGES] }
ON TABLESPACE tablespacename [, ...]
TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION]

GRANT parent_role [, ...]
TO member_role [, ...] [WITH ADMIN OPTION]

GRANT { SELECT | INSERT | ALL [PRIVILEGES] }
ON PROTOCOL protocolname
TO username
```

For more information, visit GRANT.

INSERT

Creates rows in a table.

```
INSERT INTO table [( column [, ...] )]
{DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] )
[, ...] | query}
```

For more information, visit INSERT.

LOAD

Loads or reloads a shared library file.

```
LOAD 'filename'
```

For more information, visit LOAD.

LOCK

Locks a table.

```
LOCK [TABLE] name [, ...] [IN lockmode MODE] [NOWAIT]
```

For more information, visit LOCK.

MOVE

Positions a cursor.

```
MOVE [ forward_direction {FROM | IN} ] cursorname
```

For more information, MOVE.

PREPARE

Prepares a statement for execution.

```
PREPARE name [ (datatype [, ...] ) ] AS statement
```

For more information, visit PREPARE.

REASSIGN OWNED

Changes the ownership of database objects owned by a database role.

```
REASSIGN OWNED BY old_role [, ...] TO new_role
```

For more information, visit REASSIGN OWNED.

REINDEX

Rebuilds an index.

```
REINDEX {INDEX | TABLE | DATABASE | SYSTEM} name
```

For more information, visit REINDEX.

RELEASE SAVEPOINT

Releases a savepoint.

```
RELEASE [SAVEPOINT] savepoint_name
```

For more information, visit RELEASE SAVEPOINT.

RESET

Restores a database configuration parameter to its default value.

```
RESET configuration_parameter


RESET ALL
```

For more information, visit RESET.

REVOKE

Revokes permissions for a database role.

```
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE
| REFERENCES | TRIGGER | TRUNCATE } [,...] | ALL [PRIVILEGES] }
ON [TABLE] tablename [, ...]
FROM {rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {USAGE | SELECT | UPDATE} [,...]
| ALL [PRIVILEGES] }
ON SEQUENCE sequencename [, ...]
FROM { rolename | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT
| TEMPORARY | TEMP} [,...] | ALL [PRIVILEGES] }
ON DATABASE dbname [, ...]
FROM {rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]
```

```
REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]}
ON FUNCTION funcname ( [[argmode] [argname] argtype
[, ...]] ) [, ...]
FROM {rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]}
ON LANGUAGE langname [, ...]
FROM {rolename | PUBLIC} [, ...]
[ CASCADE | RESTRICT ]

REVOKE [GRANT OPTION FOR] { {CREATE | USAGE} [,...]
| ALL [PRIVILEGES] }
ON SCHEMA schemaname [, ...]
FROM {rolename | PUBLIC} [, ...]
[CASCADE | RESTRICT]

REVOKE [GRANT OPTION FOR] { CREATE | ALL [PRIVILEGES] }
ON TABLESPACE tablespacename [, ...]
FROM { rolename | PUBLIC } [, ...]
[CASCADE | RESTRICT]

REVOKE [ADMIN OPTION FOR] parent_role [, ...]
FROM member_role [, ...]
[CASCADE | RESTRICT]
```

For more information, visit REVOKE.

ROLLBACK

Aborts the current transaction.

```
ROLLBACK [WORK | TRANSACTION]
```

For more information, visit ROLLBACK.

ROLLBACK TO SAVEPOINT

Rolls back the current transaction to a savepoint.

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] savepoint_name
```

For more information, visit ROLLBACK TO SAVEPOINT.

SAVEPOINT

Creates a savepoint within the current transaction.

```
SAVEPOINT savepoint_name
```

For more information,visit SAVEPOINT.

SELECT

Retrieves rows from a table or view.

```
[ WITH with_query [, ...] ]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
* | expression [[AS] output_name] [, ...]
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
[WINDOW window_name AS (window_specification)]
[{UNION | INTERSECT | EXCEPT} [ALL] select]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]
[LIMIT {count | ALL}]
[OFFSET start]
[FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT] [...]]
```

For more information, visit SELECT.

SELECT INTO

Creates a table from the results of a query.

```
[ WITH with_query [, ...] ]
SELECT [ALL | DISTINCT [ON ( expression [, ...] )]]
* | expression [AS output_name] [, ...]
INTO [TEMPORARY | TEMP] [TABLE] new_table
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY expression [, ...]]
[HAVING condition [, ...]]
[{UNION | INTERSECT | EXCEPT} [ALL] select]
[ORDER BY expression [ASC | DESC | USING operator] [NULLS {FIRST | LAST}] [, ...]]
[LIMIT {count | ALL}]
[OFFSET start]
[FOR {UPDATE | SHARE} [OF table_name [, ...]] [NOWAIT]
[...]]
```

For more information, visit SELECT INTO.

SET

Changes the value of a database configuration parameter.

```
SET [SESSION | LOCAL] configuration_parameter {TO | =} value |
'value' | DEFAULT}


SET [SESSION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

For more information, visit SET.

SET ROLE

Configures an identifier for the current role of the current session.

```
SET [SESSION | LOCAL] ROLE rolename


SET [SESSION | LOCAL] ROLE NONE


RESET ROLE
```

For more information, visit SET ROLE.

SET SESSION AUTHORIZATION

Configures an identifier for a session role and an identifier for the current role of the current session.

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename

SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT

RESET SESSION AUTHORIZATION
```

For more information, visit SET SESSION AUTHORIZATION.

SET TRANSACTION

Configures the characteristics of the current transaction.

```
SET TRANSACTION [transaction_mode] [READ ONLY | READ WRITE]

SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode
[READ ONLY | READ WRITE]
```

For more information, visit SET TRANSACTION.

SHOW

Shows the value of a database configuration parameter.

```
SHOW configuration_parameter

SHOW ALL
```

For more information, visit SHOW.

START TRANSACTION

Starts a transaction block.

```
START TRANSACTION [SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED]
[READ WRITE | READ ONLY]
```

For more information, visit START TRANSACTION.

TRUNCATE

Clears all rows of a table.

```
TRUNCATE [TABLE] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit TRUNCATE.

UPDATE

Updates the rows of a table.

```
UPDATE [ONLY] table [[AS] alias]
SET {column = {expression | DEFAULT} |
(column [, ...]) = ({expression | DEFAULT} [, ...])} [, ...]
[FROM fromlist]
[WHERE condition | WHERE CURRENT OF cursor_name ]
```

For more information, visit UPDATE.

VACUUM

Garbage-collects and optionally analyzes a database.

```
VACUUM [FULL] [FREEZE] [VERBOSE] [table]


VACUUM [FULL] [FREEZE] [VERBOSE] ANALYZE
[table [(column [, ...] )]]
```

For more information, visit VACUUM.

VALUES

Computes a set of rows.

```
VALUES ( expression [, ...] ) [, ...]
[ORDER BY sort_expression [ASC | DESC | USING operator] [, ...]]
[LIMIT {count | ALL}] [OFFSET start]
```

For more information, visit VALUES.

3.Manage users and permissions

Manage users

When you create an instance, the system prompts you to specify an initial username and password. This initial user is the root user. After the instance is created, you can use the credentials of the root user to connect to a database on that instance. After you use the psql CLI client of PostgreSQL or Greenplum to connect to a database on your instance, you can run the `\du+` command to view the information of all the users. Example:

> **Notice** In addition to the root user, other users are also created to manage databases.

```
postgres=> \du+
List of roles
Role name | Attributes | Member of | Description
--------------+------------------------------------+-----------+---------------
root_user | | | rds_superuser
...
```

AnalyticDB for PostgreSQL does not provide a superuser, which is equivalent to the RDS_SUPERUSER role. This is the same in ApsaraDB RDS for PostgreSQL. However, you can grant the RDS_SUPERUSER role to the root user, for example, the root_user created in the preceding example. You can only check whether the root user has this role based on the user description. The root user has the following permissions:

- Creates databases and accounts and logs on to databases, but does not have the credentials of a superuser.
- Views and modifies the tables created by users other than a superuser, changes the owners of tables, and performs operations such as SELECT, UPDATE, and DELETE.
- Views connections to users other than a superuser, cancels their SQL statements, and terminates their connections.
- Executes CREATE EXTENSION and DROP EXTENSION statements to create and delete extensions.
- Creates users who have the RDS_SUPERUSER role. Example:

  ```
  CRATE ROLE root_user2 RDS_SUPERUSER LOGIN PASSWORD 'xyz';
  ```

Manage permissions

You can manage permissions at the database, schema, and table levels. For example, if you want to grant read permissions on a table to a user and revoke write permissions, execute the following statements:

```
GRANT SELECT ON TABLE t1 TO normal_user1;
REVOKE UPDATE ON TABLE t1 FROM normal_user1;
REVOKE DELETE ON TABLE t1 FROM normal_user1;
```

References

For more information, visit Managing Roles and Privileges.

4.Insert, update, delete, and truncate data

This topic describes how to insert, update, delete, and truncate data in AnalyticDB for PostgreSQL.

Insert data

An `INSERT` statement inserts one or more rows into a table. Syntax:

```
INSERT INTO table [( column [, ...] )]
{DEFAULT VALUES | VALUES ( {expression | DEFAULT} [, ...] )
[, ...] | query}
```

For more information, visit INSERT.

Examples:

Execute the following statement to insert a row into a table:

```
INSERT INTO products (name, price, product_no) VALUES ('Cheese', 9.99, 1);
```

Execute the following statement to insert multiple rows into a table:

```
INSERT INTO products (product_no, name, price) VALUES
(1, 'Cheese', 9.99),
(2, 'Bread', 1.99),
(3, 'Milk', 2.99);
```

Execute the following statement to insert data into a table by using a scalar expression:

```
INSERT INTO films SELECT * FROM tmp_films WHERE date_prod <
'2016-05-07';
```

> **Note** If you want to insert a large volume of data, we recommend that you use an external table or execute a COPY statement to obtain a higher performance than is offered by an INSERT statement.

Update data

An `UPDATE` statement updates one or more rows in a table. Syntax:

```
UPDATE [ONLY] table [[AS] alias]
SET {column = {expression | DEFAULT} |
(column [, ...]) = ({expression | DEFAULT} [, ...])} [, ...]
[FROM fromlist]
[WHERE condition | WHERE CURRENT OF cursor_name ]
```

For more information, visit UPDATE.

Limits:

- The column defined as the distribution key cannot be updated.
- The column defined as the partition key cannot be updated.
- STABLE and VOLATILE functions are not allowed.
- RETURNING clauses are not allowed.

Example:

Execute the following statement to change all rows in which the value in the price column falls in the 5 to 10 range:

```
UPDATE products SET price = 10 WHERE price = 5;
```

Delete data

A `DELETE` statement deletes one or more rows from a table. Syntax:

```
DELETE FROM [ONLY] table [[AS] alias]
[USING usinglist]
[WHERE condition | WHERE CURRENT OF cursor_name ]
```

For more information, visit DELETE.

Limits:

- STABLE and VOLATILE functions are not allowed.
- RETURNING clauses are not allowed.

Examples:

Execute the following statement to delete all rows in which the value in the price column is 10:

```
DELETE FROM products WHERE price = 10;
```

Execute the following statement to delete all rows from a table:

```
DELETE FROM products;
```

Truncate data

A `TRUNCATE` statement clears a table. Syntax:

```
TRUNCATE [TABLE] name [, ...] [CASCADE | RESTRICT]
```

For more information, visit TRUNCATE.

Example:

In the following example, the mytable table is cleared by using a TRUNCATE statement. The TRUNCATE statement does not scan the mytable table. Therefore, it does not process the child tables of the mytable table or the rewrite rules specified by ON DELETE clauses.

```
TRUNCATE mytable;
```

5.Use INSERT ON CONFLICT to overwrite data

This topic describes how to overwrite data in AnalyticDB for PostgreSQL.

The INSERT ON CONFLICT statement allows you to update an existing row that contains a primary key when you execute the INSERT statement to insert a new row that contains the same primary key. This feature is also known as UPSERT or INSERT OVERWRITE. It is similar to the REPLACE INTO statement of MySQL.

This feature is supported in AnalyticDB for PostgreSQL 6.0 and not supported in AnalyticDB for PostgreSQL 4.3.

SQL syntax

The overwrite syntax is based on the INSERT statement described as follows:

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
{ DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
[ ON CONFLICT [ conflict_target ] conflict_action ]
[ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]

The valid value of conflict_target:

( { index_column_name | ( index_expression ) } [ COLLATE collation ] [ opclass ] [, ...] )

Valid values of conflict_action:

DO NOTHING
DO UPDATE SET { column_name = { expression | DEFAULT } |
( column_name [, ...] ) = ( { expression | DEFAULT } [, ...] )
} [, ...]
[ WHERE condition ]
```

The overwrite feature adds an ON CONFLICT clause to the common INSERT syntax. This clause contains the following parts:

- conflict_target: specifies the columns on which the primary keys conflict. If you specify conflict_action as DO NOTHING, you can omit conflict_target. If conflict_action is a DO UPDATE clause, you must specify the primary key columns or unique index columns for conflict_target.
- conflict_action: specifies the operations to be performed when the primary key conflict occurs. You can set conflict_action to DO NOTHING or DO UPDATE. DO NOTHING indicates to discard the data to be inserted and retain the original data. DO UPDATE indicates to overwrite the original data and execute the UPDATE statements.

Data to be inserted is stored in a virtual table named EXCLUDED. You can use the DO UPDATE SET clause to reference columns in the EXCLUDED table. For example, a table named tbl contains a primary key column named pri_key and a non-primary key column named col_name. If you want to overwrite an existing col_name value, you can execute the following statements:

```
insert into tbl values (0, 1), (2, 3), (4, 5)
on conflict (pri_key) do update set tbl.col_name = excluded.col_name;
```

The statement creates a virtual table named EXCLUDED that contains three rows and two columns and stores the inserted data (0, 1), (2, 3), and (4, 5). You can use excluded.col_name to reference columns in the table.

Limits

- Only AnalyticDB for PostgreSQL 6.0 supports the overwrite feature. AnalyticDB for PostgreSQL 4.3 does not support this feature.
- The table whose data is to be overwritten must be a row store table. The table cannot be a column store table because column store tables do not support unique indexes.
- The table cannot be a partition table.
- You cannot update distribution columns or primary key columns in the UPDATE SET clause.
- You cannot execute subqueries in the UPDATE WHERE clause.
- The table cannot be an updatable view.
- You cannot insert multiple data records for a primary key in an INSERT statement. This is a universal limit based on the standard SQL syntax.

Examples

- Basic usage

Execute the following statement to create a table named t1 that contains four columns. The a column is the primary key column.

```
create table t1 (a int primary key, b int, c int, d int default 0);
```

Insert a row whose primary key is 0:

```
insert into t1 values (0,0,0,0);


select * from t1;
a | b | c | d
---+---+---+---
0 | 0 | 0 | 0
(1 row)
```

Insert a new row whose primary key is also 0. The following error message is displayed.

```
insert into t1 values (0,1,1,1);

ERROR: duplicate key value violates unique constraint "t1_pkey"
DETAIL: Key (a)=(0) already exists.
```

However, operations are required instead of an error message in some scenarios. The original data must be retained or updated after the primary key conflict occurs. You can use the overwrite feature to perform these operations.

Execute the following statement to retain the original data and discard the new data:

```
-- Use the ON CONFLICT DO NOTHING clause.

insert into t1 values (0,1,1,1) on conflict do nothing;

select * from t1;
a | b | c | d
---+---+---+---
0 | 0 | 0 | 0
(1 row)
```

Execute the following statement to overwrite the original data with the new data:

```
-- Use the ON CONFLICT DO UPDATE clause.

insert into t1 values (0,2,2,2) on conflict (a) do update set (b, c, d) = (excluded.b, excluded.c, excluded.
d);

select * from t1;
a | b | c | d
---+---+---+---
0 | 2 | 2 | 2
(1 row)
```

The ON CONFLICT DO NOTHING and DO UPDATE clauses enable you to overwrite data as needed. The EXCLUDED table indicates the virtual table that stores the (0,2,2,2) row.

The preceding statement can also be written in the following format:

```
insert into t1 values (0,2,2,2) on conflict (a) do update set b = excluded.b, c = excluded.c, d = excluded.
d;
```

The overwrite feature can also be used in the following ways:

- Execute the following statement to overwrite some columns of the original data with the new data:

```
-- Overwrite the c column with the value of excluded.c, which indicates the c column of the new data.

insert into t1 values (0,0,3,0) on conflict (a) do update set c = excluded.c;

select * from t1;
a | b | c | d
---+---+---+---
0 | 2 | 3 | 2
(1 row)
```

- Execute the following statement to update some columns of the original data:

```
-- Add 1 to the original d column value.

insert into t1 values (0,0,3,0) on conflict (a) do update set d = t1.d + 1;

select * from t1;
a | b | c | d
---+---+---+---
0 | 2 | 3 | 3
(1 row)
```

- Execute the following statement to set column values to default values:

```
-- Set the d column value to the default value 0 as defined in the CREATE TABLE statement.

insert into t1 values (0,0,3,0) on conflict (a) do update set d = default;

select * from t1;
a | b | c | d
---+---+---+---
0 | 2 | 3 | 0
(1 row)
```

- Execute the following statements to insert multiple rows:

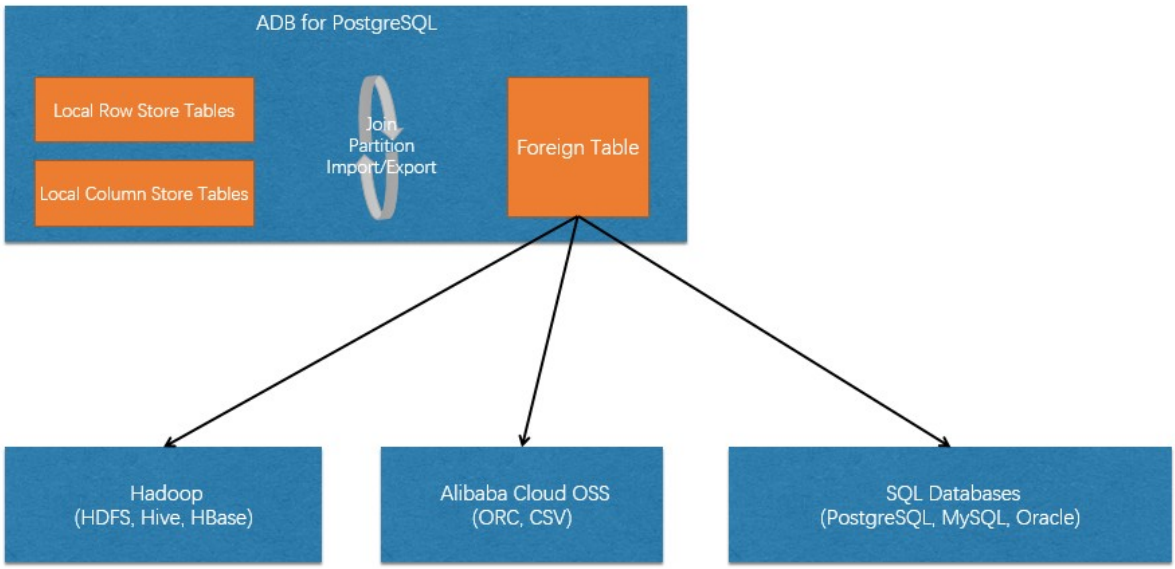
```
-- Insert two rows. The original row with the primary key 0 is retained because a primary key conflict o
ccurs and the row with the primary key 1 is inserted to the table.

insert into t1 values (0,0,0,0), (1,1,1,1) on conflict do nothing;

select * from t1;
a | b | c | d
---+---+---+---
0 | 2 | 3 | 0
1 | 1 | 1 | 1
(2 rows)

-- Insert two rows. The new row with the primary key 0 overwrites the original row because a primary
key conflict occurs and the row with the primary key 2 is inserted to the table.

insert into t1 values (0,0,0,0), (2,2,2,2) on conflict (a) do update set (b, c, d) = (excluded.b, excluded.c, e
xcluded.d);

select * from t1;
a | b | c | d
---+---+---+---
0 | 0 | 0 | 0
1 | 1 | 1 | 1
2 | 2 | 2 | 2
(3 rows)
```

- Execute the following statements to insert data obtained by subqueries to merge data in two tables or perform complex INSERT INTO and SELECT statements:

```
create table t2 (like t1);
insert into t2 values (2,22,22,22),(3,33,33,33);

-- Insert rows from the t2 table to the t1 table. If a primary key conflict occurs, the rows that have the same primary keys in t2 overwrite the rows in t1.

insert into t1 select * from t2 on conflict (a) do update set (b, c, d) = (excluded.b, excluded.c, excluded.d);

select * from t1;
a | b | c | d
---+----+----+----
0 | 0 | 0 | 0
1 | 1 | 1 | 1
2 | 22 | 22 | 22
3 | 33 | 33 | 33
(4 rows)
```

6.Use OSS foreign tables to access OSS data

OSS foreigntables are designed based on the PostgreSQL Foreign Data Wrapper (FDW) framework to access OSS data for analysis.



OSS foreign tables allow you to perform the following operations:

- Import OSS data to local row-oriented and column-oriented tables for accelerated analysis.
- Directly query and analyze huge amounts of OSS data.
- Join OSS foreign tables and local tables for data analysis

OSS foreign tables support gzip-compressed files in the ORC and CSV formats and can be partitioned based on one or more columns for partition filtering.

OSS data may come from business applications, log archives of Alibaba Cloud Log Service, and ETL operations of Data Lake Analysis.

Use an OSS foreign table

The procedure of using an OSS foreign table can be simplified as follows:

- Create a *user mapping* to a foreign server. For more information, visit CREATE USER MAPPING.
- Create an *OSS foreign server*. For more information, visit CREATE SERVER.
- Create an OSS *foreign table*. For more information, visit CREATE FOREIGN TABLE.

For example, you can use the ossutil Overview to view the following information about a TPC-H lineitem table in an OSS bucket.

```
[adbpgadmin@localhost]$ ossutil ls oss://adbpg-tpch/data/tpch_data_10x/lineitem.tbl
LastModifiedTime Size(B) StorageClass ETAG ObjectName
2020-03-12 09:29:48 +0800 CST 144144997 Standard 1F426F2FFC70A0262D2D69183BC3A0BD-57 oss://ad
bpg-tpch/data/tpch_data_10x/lineitem.tbl.1
2020-03-12 09:29:58 +0800 CST 145177420 Standard CFE2CFF1C8059547DC9F1711E77F74DD-57 oss://adb
pg-tpch/data/tpch_data_10x/lineitem.tbl.10
2020-03-10 21:23:24 +0800 CST 145355168 Standard 35C6227D1C29F1236A92A4D5D7922625-57 oss://adb
pg-tpch/data/tpch_data_10x/lineitem.tbl.11
... ...
```

In the preceding information,

- **adbpg-tpch** is the name of the OSS bucket.
- **data/tpch_data_10x/...** is the path of the file relative to the OSS bucket.

The following section describes how to create and use an OSS foreign table in detail.

1. Create an OSS server

To create an OSS server, you must specify <oss endpoint> to access the OSS server.

1.1 Example

```
CREATE SERVER oss_serv -- The name of the OSS server.
FOREIGN DATA WRAPPER oss_fdw
OPTIONS (
endpoint '<Oss endpoint>', -- The endpoint of the OSS server.
bucket '<Oss bucket>' -- The bucket where the data file is located.
);
```

1.2 Syntax

```
-- Create an OSS server.
CREATE SERVER server_name [ TYPE 'server_type' ] [ VERSION 'server_version' ]
FOREIGN DATA WRAPPER fdw_name
[ OPTIONS ( option 'value' [, ... ] ) ]

-- Delete an OSS server.
DROP SERVER [ IF EXISTS ] name [ CASCADE | RESTRICT ]
```

For more information about options in the OPTIONS clause, see 1.3 Parameter options. For more information about how to create an OSS server, visit CREATE SERVER.

1.3 Parameter options

Option	Type	Unit	Required	Default value	Description

endpoint	String		Yes		The endpoint corresponding to the OSS region. **Note:** If you access your AnalyticDB for PostgreSQL instance from an ECS instance, we recommend that you use an internal endpoint containing the "internal" keyword to avoid incurring public traffic.

bucket	String		No		Specifies the bucket where the data file is located. You must use OSS to create the bucket before data import. **Note:** You must set bucket for either an OSS server or an OSS table. If you set bucket for both, the bucket value for the OSS table overwrites that for the OSS server.
Note: • The following error-tolerance parameters can be used when you access OSS. You can retain the default values. • If you retain the default values for the following parameters, a timeout is triggered after the transmission rate remains lower than 1 KB/s for 1,500 consecutive seconds. For more information, see Error handling.					
speed_limit	Numeric	Bytes/second	No	1024	Specifies the minimum tolerable transmission rate.
speed_time	Numeric	Seconds	No	1500	Specifiesthe maximum tolerable duration for speed_limit.

connect_timeout	Numeric	Seconds	No	10	Specifies the connection timeout period.
dns_cache_timeout	Numeric	Seconds	No	60	Specifies the timeout period for DNS resolution.

2. Create a user mapping to an OSS server

After creating an OSS server, you need to create a user mapping from your AnalyticDB for PostgreSQL instance to the OSS server.

2.1 Example

```
CREATE USER MAPPING FOR PUBLIC -- Create a user mapping to the OSS server for all users. For more i
nformation, see 2.2 Syntax.
SERVER oss_serv -- Specify the OSS server that you want to access.
OPTIONS (
id '<oss access id>', -- Specify the AccessKey ID of the OSS account.
key '<oss access key>' -- Specify the AccessKey secret of the OSS account.
);
```

2.2 Syntax

```
-- Create a user mapping.
CREATE USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }
SERVER servername
[ OPTIONS ( option 'value' [, ... ] ) ]

-- Delete a user mapping.
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC }
SERVER server_name
```

Description:

- **username** The name of an existing user that is mapped to foreign server.
- **CURRENT_USER** and **USER** match the name of the current user.
- **PUBLIC** all roles, including those that might be created later.

For more information about options in the OPTIONS clause, see2.3 Parameter options. For more information about how to create a user mapping, visit CREATE USER MAPPING.

2.3 Parameter options

Option	Required	Default value	Description
id	Yes		The AccessKey ID of the OSS account.
key	Yes		The AccessKey secret of the OSS account.

3. Create an OSS foreign table

After an OSS bucket and the account that is used to access the OSS bucket are specified, you can define an OSS foreign table. OSS foreign tables support data files in multiple formats for different business scenarios.

- OSS foreign tables allow you to access uncompressed files in the CSV and TEXT formats.
- OSS foreign tables allow you to access gzip-compressed files in the CSV and TEXT formats.
- OSS foreign tables allow you to access binary files in the ORC format. For more information, see the "Data type mappings between ORC files and AnalyticDB for PostgreSQL files" section in this topic.

3.1 Example

```
-- Create an uncompressed OSS foreign table in the TEXT format.
CREATE FOREIGN TABLE oss_lineitem ( -- The name of the OSS foreign table.
l_orderkey bigint,
l_partkey bigint,
l_suppkey bigint,
l_linenumber bigint,
l_quantity double precision,
l_extendedprice double precision,
l_discount double precision,
l_tax double precision,
l_returnflag CHAR(1),
l_linestatus CHAR(1),
l_shipdate DATE,
l_commitdate DATE,
l_receiptdate DATE,
l_shipinstruct CHAR(25),
l_shipmode CHAR(10),
l_comment VARCHAR(44)
) server oss_serv -- Specify the OSS server.
options (
prefix 'data/tpch_data_10x/lineitem.tbl', -- Specify the prefix with which to match OSS files.
format 'text', -- Parse files in the TEXT format.
delimiter '|' -- Specify the column delimiter.
);
```

```
-- Create a gzip-compressed OSS foreign table in the CSV format.
CREATE FOREIGN TABLE oss_lineitem_csv_gz ( -- The name of the OSS foreign table.
l_orderkey bigint,
l_partkey bigint,
l_suppkey bigint,
l_linenumber bigint,
l_quantity double precision,
l_extendedprice double precision,
l_discount double precision,
l_tax double precision,
l_returnflag CHAR(1),
l_linestatus CHAR(1),
l_shipdate DATE,
l_commitdate DATE,
l_receiptdate DATE,
l_shipinstruct CHAR(25),
l_shipmode CHAR(10),
l_comment VARCHAR(44)
) server oss_serv -- Specify the OSS server.
options (
prefix 'data/tpch_data_10x/lineitem.tbl.gz', --Specify the prefix with which to match OSS files.
format 'csv', -- Parse files in the CSV format.
delimiter '|', -- Specify the column delimiter.
filetype 'gzip' -- Specify the gzip-compressed file.
);

-- Create an OSS foreign table in the ORC format.
CREATE FOREIGN TABLE oss_lineitem_orc ( -- The name of the OSS foreign table.
l_orderkey bigint,
l_partkey bigint,
l_suppkey bigint,
l_linenumber bigint,
l_quantity double precision,
l_extendedprice double precision,
l_discount double precision,
l_tax double precision,
l_returnflag CHAR(1),
l_linestatus CHAR(1),
l_shipdate DATE,
l_commitdate DATE
```

```
l_commitdate DATE,
l_receiptdate DATE,
l_shipinstruct CHAR(25),
l_shipmode CHAR(10),
l_comment VARCHAR(44)
) server oss_serv -- Specify the OSS server.
options (
prefix 'data/tpch_orc_data_10x/lineitem.orc', -- Specify the prefix with which to match OSS files.
format 'orc' -- Parse files in the ORC format.
);
```

Note:

After you create an OSS foreign table, you can use one of the following statements to check whether OSS objects that match the table meet expectations:

- Statement 1: `explain verbose select * from <OSS foreign table>;`
- Statement 2: `select * from get_oss_table_meta('<OSS foreign table>');`

3.2 Syntax

```
-- Create an OSS foreign table.
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name ( [
column_name data_type [ OPTIONS ( option 'value' [, ... ] ) ] [ COLLATE collation ] [ column_constraint [ ...
] ]
[, ... ]
] )
SERVER server_name
[ OPTIONS ( option 'value' [, ... ] ) ]

-- Delete an OSS foreign table.
DROP FOREIGN TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

For more information about options in the OPTIONS clause, see3.3 Parameter options. For more information about how to create a foreign table, visit CREATE FOREIGN TABLE.

3.3 Parameter options

3.3.1 Common options

Option	Type	Unit	Required	Default value	Description

filepath	String		Yes. Select one of the three options. **Note:** The three options all specify the paths relative to the OSS bucket.		Matches a single file.
prefix	String				Matches multiple files by prefix.
dir	String				Matches a list of files and subfolders. The dir value must end with a forward slash (/).
bucket	String		No		You must set bucket for either an OSS server or an OSS table. If you set bucket for both, the bucket value for the OSS table overwrites that for the OSS server.
format	String		Yes		Specifies the file format. Valid values: • csv • text • orc

3.3.2 Options for CSV and TEXT files

Note: The following options are valid only for CSV and TEXT files and invalid for ORC files.

Option	Type	Unit	Required	Default value	Description

filetype	String		No	plain	Valid values: • plain: only reads the raw binary data. • gzip: reads the raw binary data and decompresses the GZIP package.
log_errors	Boolean		No	false	Specifies whether to record errors in log files. For more information, see 3.4 Fault tolerance mechanism.
					Specifies the maximum allowable number of error lines before the execution is aborted. If the value contains a percent sign (%), it indicates the percentage of error lines. Otherwise, the value indicates the number of error lines. Example: • segment_reject_limit = '10

segment_reject_limit	Numeric		No		indicates that execution is aborted when the number of error lines on a compute node exceeds 10. • segment_reject_limit = '10 indicates that execution is aborted when the number of error lines on a compute node is more than 10% of the processed lines. For more information, see 3.4 Fault tolerance mechanism.
The following table describes the formatting options. For more information, visit COPY					
header	Boolean		No	false	Specifies whether the source file contains the header column. • This option is only valid for CSV files.

delimiter	String		No	Default delimiter for **TEXT** files: tab. Default delimiter for **CSV** files: comma (,).	The column delimiter • Only single-byte characters are allowed.
quote	String		No	Double quotation mark (").	The column quotation • This option is only valid for CSV files. • Only single-byte characters are allowed.
escape	String		No	The value is the same as the quote value by default.	Specifies the character that appears before the data character that matches the quote value. • Only single-byte characters are allowed. • This option is only valid for CSV files.
null	String			Default delimiter for **TEXT** files: \N. Default delimiter for **CSV** files: spaces that are not enclosed in quotes.	Specifies the null string for a file.

encoding	String		No	If you do not specify the option, the encoding type on the client is used by default.	Specifies the encoding format of local data files.
force_not_null	Boolean		No	false	If the option is set to true, the values of specified columns are matched against the null string.
force_null	Boolean		No	false	• If this option is set to true, the column values that match the null string are returned as NULL even if the values are quoted. • Without this option, only unquoted values matching the null string are returned as NULL.

3.4 Fault tolerance mechanism

When creating an OSS foreign table, you can set the **log_errors** and **segment_reject_limit** options to avoid unexpected exits due to error lines of raw data during a scan of OSS foreign tables. In the preceding information,

- "***log_errors***" indicates whether to record information of the error lines.
- "***segment_reject_limit***" indicates the fault tolerance ratio, namely the percentage of error lines in all parsed lines.

Note: OSS foreign tables in the ORC format do not support fault tolerance.

1. Create a FDW-based OSS foreign table that supports fault tolerance.

```
create foreign table oss_error_sales (id int, value float8, x text)
server oss_serv
options (log_errors 'true', -- Record the information of the error lines.
segment_reject_limit '10', -- The number of error lines can be up to 10. Otherwise, the system retur
ns an error and exits.
dir 'error_sales/', -- Specify the OSS directory that the foreign table matches.
format 'csv', -- Parse files in the CSV format.
encoding 'utf8'); -- Specify the encoding format.
```

2. Scan the foreign table.

   Three lines of error records are added to an OSS file to demonstrate the effect of fault tolerance.

```
postgres=# select * from oss_error_sales ;
NOTICE: found 3 data formatting errors (3 or more input rows), rejected related input data
id | value | x
----+---------------+-----------
1 | 0.1102213212 | abcdefg
1 | 0.1102213212 | abcdefg
2 | 0.92983182312 | mmsmda123
3 | 0.1123123 | abbs
1 | 0.1102213212 | abcdefg
2 | 0.92983182312 | mmsmda123
3 | 0.1123123 | abbs
1 | 0.1102213212 | abcdefg
1 | 0.1102213212 | abcdefg
2 | 0.92983182312 | mmsmda123
3 | 0.1123123 | abbs
3 | 0.1123123 | abdsa
1 | 0.1102213212 | abcdefg
2 | 0.92983182312 | mmsmda123
3 | 0.1123123 | abbs
1 | 0.1102213212 | abcdefg
2 | 0.92983182312 | mmsmda123
3 | 0.1123123 | abbs
(18 rows)
```

3. View the log of error lines.

```
postgres=# select * from gp_read_error_log('oss_error_sales');
cmdtime | relname | filename | linenum | bytenum | errmsg | rawdata | rawbytes
-------------------------------+-----------------+-------------------------+---------+---------+------------------
----------------------------------------+---------+----------
2020-04-22 19:37:35.21125+08 | oss_error_sales | error_sales/sales.2.csv | 2 | | invalid byte sequenc
e for encoding "UTF8": 0xed 0xab 0xad | | \x
2020-04-22 19:37:35.21125+08 | oss_error_sales | error_sales/sales.2.csv | 3 | | invalid byte sequenc
e for encoding "UTF8": 0xed 0xab 0xad | | \x
2020-04-22 19:37:35.21125+08 | oss_error_sales | error_sales/sales.3.csv | 2 | | invalid byte sequenc
e for encoding "UTF8": 0xed 0xab 0xad | | \x
(3 rows)
```

4. Delete the log of error lines.

```
postgres=# select gp_truncate_error_log('oss_error_sales');
gp_truncate_error_log
-----------------------
t
(1 row)
```

4. Use an OSS foreign table

4.1 Import OSS data to local tables

Perform the following steps to import data:

1. Distribute data evenly among multiple OSS files.

   Note:

   - All compute nodes of an AnalyticDB for PostgreSQL instance read data in parallel from the data files stored in OSS by using a round-robin algorithm.
   - We recommend that you use the same data encoding formats for data files and databases to simplify the encoding process and maximize efficiency. The default database encoding format is UTF-8.
   - Multiple CSV or TEXT files can be read in parallel. The default number of files to read in parallel is 4.
   - To maximize reading efficiency, we recommend that you set the number of files as an integer multiple of the number of data nodes. The number of data nodes is the product of the number of compute nodes and the number of cores per compute node.
2. Create an OSS foreign table for each database in your AnalyticDB for PostgreSQL instance.
3. Execute the following statement to import data in parallel:

```
-- INSERT statement
INSERT INTO <Local destination table> SELECT * FROM <OSS foreign table>;

-- CREATE TABLE AS statement
CREATE TABLE <Local destination table> AS SELECT * FROM <OSS foreign table>;
```

- **Example 1** - Use the INSERT statement to import the oss_lineitem_orc data to a local AOCS table

```
-- Create a local AOCS table.
CREATE TABLE aocs_lineitem (
l_orderkey bigint,
l_partkey bigint,
l_suppkey bigint,
l_linenumber bigint,
l_quantity double precision,
l_extendedprice double precision,
l_discount double precision,
l_tax double precision,
l_returnflag CHAR(1),
l_linestatus CHAR(1),
l_shipdate DATE,
l_commitdate DATE,
l_receiptdate DATE,
l_shipinstruct CHAR(25),
l_shipmode CHAR(10),
l_comment VARCHAR(44)
) WITH (APPENDONLY=TRUE, ORIENTATION=COLUMN, COMPRESSTYPE=ZSTD, COMPRESSLEVEL=5);

-- Import the oss_lineitem_orc data to the local AOCS table.
INSERT INTO aocs_lineitem SELECT * FROM oss_lineitem_orc;
```

- **Example 2** - Use the CREATE TABLE AS statement to import the oss_lineitem_orc data to a local heap table

```
create table heap_lineitem as select * from oss_lineitem_orc distributed by (l_orderkey);
```

4.2 Query and analyze OSS data

You can query an OSS foreign table in the same way as a local table. Common query scenarios are as follows:

- Filtering by key-value pair

```
select * from oss_lineitem_orc where l_orderkey = 14062498;
```

- Aggregation query

```
select count(*) from oss_lineitem_orc where l_orderkey = 14062498;
```

- Filtering and grouping followed by the LIMIT clause

```
select l_partkey, sum(l_suppkey)
from oss_lineitem_orc
group by l_partkey
order by l_partkey
limit 10;
```

4.3 Join OSS foreign tables and local tables for data analytics

- Example - Join local AOCS table aocs_lineitem and OSS foreign tables for a TPC-H Q3 query

```
select l_orderkey,
sum(l_extendedprice * (1 - l_discount)) as revenue,
o_orderdate,
o_shippriority
from oss_customer, -- OSS foreign table
oss_orders, -- OSS foreign table
aocs_lineitem -- Local AOCS table
where c_mktsegment = 'furniture'
and c_custkey = o_custkey
and l_orderkey = o_orderkey
and o_orderdate < date '1995-03-29'
and l_shipdate > date '1995-03-29'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
limit 10;
```

5. Use a partitioned OSS foreign table

AnalyticDB for PostgreSQL has added support for partitions on a basis of OSS foreign tables. Using partition columns in a WHERE condition can significantly reduce the amount of data obtained from remote servers.

The partition function of FDW-based OSS foreign tables in AnalyticDB for PostgreSQL has certain requirements for file organization. The partition data of a partitioned OSS foreign table must be located within the `oss://bucket/partcol1=partval1/partcol2=partval2/` directory. In the preceding directory, partcol1 and partcol2 indicate partition columns. partval1 and partval2 indicate the corresponding values of the partition columns.

5.1 Example

The following SQL statements create the userlogin foreign table that has two partitions. The paths of the two partitions in OSS are `oss://bucket/userlogin/month=201812/` and `oss://bucket/userlogin/month=201901/`.

```
CREATE FOREIGN TABLE userlogin (
uid integer,
name character varying,
source integer,
logindate timestamp without time zone,
month int
) SERVER oss_serv OPTIONS (
dir 'userlogin/',
format 'text'
)
PARTITION BY LIST (month)
(
VALUES ('201812'),
VALUES ('201901')
)
```

5.2 Syntax

FDW-based OSS foreign tables in AnalyticDB for PostgreSQL and standard partitioned tables use the same syntax to perform partitioning. For more information about the partitioning syntax of standard partitioned tables, see Table partitioning. FDW-based OSS foreign tables in AnalyticDB for PostgreSQL support only list partitions.

5.3 Usage scenario

Partitioned foreign tables are commonly used to access logs delivered by Log Service. For more information, see Use FDW-based OSS foreign tables to access logs delivered by Log Service. You can use a similar method to organize directories when writing data from business apps to OSS, and define partitions for foreign tables.

Collect statistics of foreign tables

Data of OSS foreign tables is stored in OSS. By default, the statistics of foreign tables is not collected. When the statistics does not exist or is stale, a query optimizer may generate inefficient query plans for complicated queries such as JOIN operations. You can use the ANALYZE statement to update statistics. The execution process is as follows:

```
-- Execute the EXPLAIN statement to view the query plan before the ANALYZE statement is executed.
EXPLAIN <Table name>;

-- Use the ANALYZE statement to collect statistics of an OSS foreign table.
ANALYZE <Table name>;

-- Use the EXPLAIN statement to view the query plan after the ANALYZE statement is executed.
EXPLAIN <Table name>;
```

Perform the mapping between the number of OSS foreign tables and the number of compute nodes

- All compute nodes of an AnalyticDB for PostgreSQL instance read data in parallel from the data files stored in OSS by using a round-robin algorithm.
- We recommend that you use the same data encoding formats for data files and databases to simplify the encoding process and maximize efficiency. The default database encoding format is UTF-8.
- Multiple CSV or TEXT files can be read in parallel. The default number of files to read in parallel is 4.
- To maximize reading efficiency, we recommend that you set the number of files as an integer multiple of the number of data nodes. The number of data nodes is the product of the number of compute nodes and the number of cores per compute node.

View query plans

Execute the following statement to view the query plan of an OSS foreign table:

```
EXPLAIN SELECT COUNT(*) FROM oss_lineitem_orc WHERE l_orderkey > 14062498;
```

Note:

- You can execute the EXPLAIN VERBOSE statement to view more information.

View OSS file information

You can execute the following statement to obtain file information of a specified OSS foreign table:

```
SELECT * FROM get_oss_table_meta('<OSS FOREIGN TABLE>');
```

Data type mappings between ORC files and AnalyticDB for PostgreSQL files

The ORC files support 18 data types, among whichMap, Struct, and Union have no mappings in AnalyticDB for PostgreSQL. ORC files of the LIST type can only be converted into one-dimensional arrays in AnalyticDB for PostgreSQL.

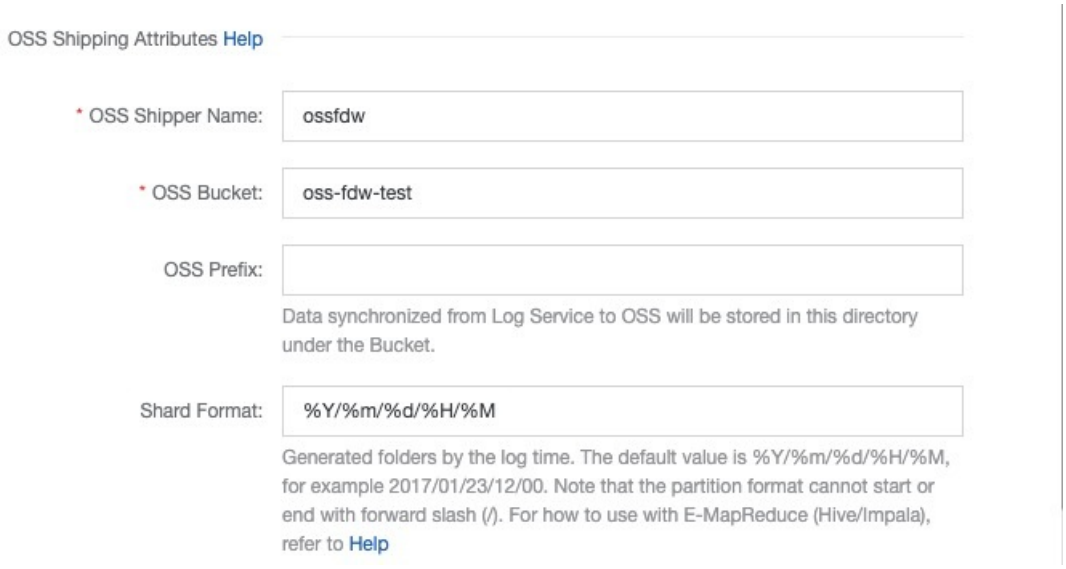
The following table lists the mappings between ORC files and AnalyticDB for PostgreSQL files.

AnalyticDB for PostgreSQL data type	ORC data type
BOOL	BOOLEAN
INT2	SHORT
INT4	INT
INT8	LONG
FLOAT4	FLOAT
FLOAT8	DOUBLE
NUMERIC	DECIMAL
CHAR	CHAR
TEXT	STRING
BYTEA	BINARY
TIMESTAMP	TIMESTAMP
DATE	DATE
INT2[]	LIST(SHORT)
INT4[]	LIST(INT)
INT8[]	LIST(LONG)
FLOAT4[]	LIST(FLOAT)
FLOAT8[]	LIST(DOUBLE)

Use FDW-based OSS foreign tables to access logs delivered by Log Service

Partitions of FDW-based OSS foreign tables in AnalyticDB for PostgreSQL are more widely used together with Log Service. Log Service is an end-to-end logging service developed by Alibaba Cloud and is widely used in big data scenarios. For more information about Log Service, see What is Log Service?

If you want to create a partitioned OSS foreign table based on the data delivered from Log Service to OSS, you must set Shard Format in the OSS LogShipper dialog box as shown in the following figure. For more information about the parameters, see Ship logs to OSS.



If the parameters are configured as shown in the preceding figure, logs generated within the same month are stored in a single OSS directory. The configuration generates the following directory layout in OSS:

```
oss://oss-fdw-test/adbpgossfdw
├── date=202002
│   ├── userlogin_1585617629106546791_647504382.csv
│   └── userlogin_1585617849232201154_647507440.csv
└── date=202003
    └── userlogin_1585617944247047796_647508762.csv
```

Then, create the following partitioned OSS foreign table based on the columns contained in the files delivered by Log Service:

```
CREATE FOREIGN TABLE userlogin (
uid integer,
name character varying,
source integer,
logindate timestamp without time zone,
"date" int
) SERVER oss_serv OPTIONS (
dir 'adbpgossfdw/',
format 'text'
)
PARTITION BY LIST ("date")
(
VALUES ('202002'),
VALUES ('202003')
)
```

Finally, implement required analysis logic based on the partitioned foreign table. For example, you can analyze all user logons in February 2020.

```
adbpg=# explain select uid, count(uid) from userlogin where "date" = 202002 group by uid;
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
Gather Motion 3:1 (slice2; segments: 3) (cost=5135.10..5145.10 rows=1000 width=12)
-> HashAggregate (cost=5135.10..5145.10 rows=334 width=12)
Group Key: userlogin_1_prt_1.uid
-> Redistribute Motion 3:3 (slice1; segments: 3) (cost=5100.10..5120.10 rows=334 width=12)
Hash Key: userlogin_1_prt_1.uid
-> HashAggregate (cost=5100.10..5100.10 rows=334 width=12)
Group Key: userlogin_1_prt_1.uid
-> Append (cost=0.00..100.10 rows=333334 width=4)
-> Foreign Scan on userlogin_1_prt_1 (cost=0.00..100.10 rows=333334 width=4)
Filter: (date = 202002)
Oss Url: endpoint=oss-cn-hangzhou-zmf-internal.aliyuncs.com bucket=adbpg-regress dir=adbpgossfd
w/date=202002/ filetype=plain|text
Oss Parallel (Max 4) Get: total 0 file(s) with 0 bytes byte(s).
Optimizer: Postgres query optimizer
(13 rows)
```

In the preceding statements, the FDW-based OSS foreign table reads only data generated in February 2020, rather than data generated in March 2020 to reduce the amount of data to read and maximize query efficiency.

Troubleshooting

During a scan of an OSS foreign table, the following error message may be displayed:

"oss server returned error: StatusCode=..., ErrorCode=..., ErrorMessage="...", RequestId=..."

Where:

- StatusCode: the HTTP status code corresponding to the error.
- ErrorCode: the error code returned by OSS.
- ErrorMessage: the error message returned by OSS.
- RequestId: the UUID that identifies the request. If you cannot solve a problem, provide the RequestId to seek assistance from OSS development engineers.

For more information about error types, seeOSS error response.

Differences between OSS foreign tables and OSS external tables

- AnalyticDB for PostgreSQL allows you to use OSS external tables for data import and export. However, OSS external tables cannot meet requirements for analysis of large amounts of data.
- OSS foreign tables are developed based on the PostgreSQL Foreign Data Wrapper (FDW) framework. OSS foreign tables support gzip-compressed files in the CSV and TEXT formats and can be partitioned based on one or more columns. OSS foreign tables allow you to collect the statistics of foreign table so that the optimizer can generate an optimal query plan.
- Foreign tables are superior to external tables in terms of performance, functionality, and stability. Therefore, the Greenplum community plans to replace external tables with foreign tables.

References

- Get started with Object Storage Service
- OSS domain names
- OSS SDK troubleshooting
- OSS respone to common errors

7.Transaction management

AnalyticDB for PostgreSQL supports standard attributes of database transactions and three isolation levels. These attributes include atomicity, consistency, isolation, and durability, which are collectively referred to as ACID. AnalyticDB for PostgreSQL uses a distributed massively parallel processing (MPP) architecture to horizontally scale nodes and ensure transaction consistency between nodes. This topic describes the transaction isolation levels and transaction-related operations supported by AnalyticDB for PostgreSQL.

Isolation levels

AnalyticDB for PostgreSQL provides the following three transaction isolation levels:

- READ UNCOMMITTED: follows standard SQL syntax. However, this isolation level is implemented the same as the READ COMMITED isolation level in AnalyticDB for PostgreSQL.
- READ COMMITTED: follows standard SQL syntax and is implemented the same as the READ COMMITED isolation level in AnalyticDB for PostgreSQL.
- SERIALIZABLE: follows standard SQL syntax. However, this isolation level is implemented the same as the REPEATABLE READ isolation level in AnalyticDB for PostgreSQL.

Example:

Execute the following statements to start a transaction block with the SERIALIZABLE isolation level:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

SQL statements supported

AnalyticDB for PostgreSQL provides the following SQL statements for you to manage transactions:

- BEGIN and START: each start a transaction block.
- END and COMMIT: each commit a transaction.
- ROLLBACK: rolls back a transaction with no changes retained.
- SAVEPOINT: creates a savepoint within a transaction. You can revoke the SQL statements executed after the savepoint was created.
- ROLLBACK TO SAVEPOINT: rolls back a transaction to a savepoint.
- RELEASE SAVEPOINT: releases a savepoint from a transaction.

Examples:

Execute the following statements to create a savepoint in a transaction and revoke the SQL statements executed after the savepoint is created:

```
BEGIN;
INSERT INTO table1 VALUES (1);
SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (2);
ROLLBACK TO SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (3);
COMMIT;
```

In this example, the values 1 and 3 are inserted, but the value 2 is not.

Execute the following statements to create a savepoint in a transaction and then release the savepoint:

```
BEGIN;
INSERT INTO table1 VALUES (3);
SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (4);
RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

In this example, the values 3 and 4 are inserted.

8.Manage JSON and JSON-B data

JavaScript Object Notation (JSON) data types are widely used on the Internet and Internet of Things (IoT). For more information about the protocols used for JSON, visit Introducing JSON. AnalyticDB for PostgreSQL supports JSON data types. AnalyticDB for PostgreSQL V6.0 also supports JSON-B data types. This topic describes how to manage JSON and JSON-B data, including:

- Differences and similarities between JSON and JSON-B
- JSON input and output syntax
- JSON operators
- JSON-B operators
- JSON creation functions
- JSON processing functions
- Create a JSON-B function index

Differences and similarities between JSON and JSON-B

JSON and JSON-B data types are similar in usage but different in storage methods. JSON data is stored as an exact copy of the input text, while JSON-B data is stored in a binary format. The JSON-B data type is more efficient and processes faster than the JSON data type. Also, the JSON-B data type supports indexing. Therefore, the JSON-B data type is preferred for AnalyticDB for PostgreSQL V6.0.

JSON input and output syntax

For more information, visit RFC 7159.

A JSON value must be an object, array, number, string, or one of the following literal names in lowercase: true, null, and false. The following statements are valid JSON expressions:

```
-- A simple scalar or value
-- A simple value must be a number, a string enclosed in a pair of quotation marks, or a literal name (tr
ue, false, or null).
SELECT '5'::json;

-- An array of zero or more elements (The elements can be of the same type or different types.)
SELECT '[1, 2, "foo", null]'::json;

-- An object that contains key-value pairs
-- The key in a key-value pair of an object must be a string enclosed in a pair of quotation marks.
SELECT '{"bar": "baz", "balance": 7.77, "active": false}'::json;

-- An array or object nested in each other
SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::json;
```

The preceding JSON values can all be converted into JSON-B values. Example:

```
-- A simple scalar or value in the JSON-B format
SELECT '5'::jsonb;
```

JSON operators

The following table describes the operators available for use with JSON and JSON-B data types.

Operator	Right operand type	Description	Example	Result
->	int	Obtains a JSON array element indexed from zero.	`'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2`	`{"c":"baz"}`
->	text	Obtains a JSON object field based on a key.	`'{"a":{"b":"foo"}}'::json->'a'`	`{"b":"foo"}`
->>	int	Obtains a JSON array element as text.	`'[1,2,3]'::json->>2`	`3`
->>	text	Obtains a JSON object field as text.	`'{"a":1,"b":2}'::json->>'b'`	`2`
#>	text[]	Obtains a JSON object from a specified path.	`'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'`	`{"c": "foo"}`
#>>	text[]	Obtains a JSON object as text from a specified path.	`'{"a":[1,2,3], "b":[4,5,6]}'::json#>>'{a,2}'`	`3`

The following table describes the operators available for use with JSON-B data types.

JSON-B operators

The following table describes the operators available for use with JSON-B data types.

Operator	Right operand type	Description	Example
=	jsonb	Specifies whether two JSON values are the same.	`'[1,2]'::jsonb= '[1,2]'::jsonb`

Operator	Right operand type	Description	Example
@>	jsonb	Specifies whether the left JSON value contains the right JSON value.	'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb
<@	jsonb	Specifies whether the left JSON value is contained within the right JSON value.	'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb
?	text	Specifies whether the string exists as a key or as a string within the JSON value	'{"a":1, "b":2}'::jsonb ? 'b'
? |	text[]	Specifies whether any of the right array strings exist as keys or as strings within the JSON value.	'{"a":1, "b":2, "c":3}'::jsonb ? | array['b', 'c']
? &	text[]	Specifies whether all of the right array strings exist as keys or as strings within the JSON value.	'["a", "b"]'::jsonb ? & array['a', 'b']

JSON creation functions

The following table describes the functions used to create JSON values.

Function	Description	Example	Result
to_json (anyelement)	Returns a value as a valid JSON object. Arrays and composites are converted recursively to arrays and objects. If a cast function is provided, that cast function is invoked to convert the input value into a JSON object. Otherwise, a scalar value is produced. If the scalar value is not a number, Boolean value, or null, it is represented by JSON text with quotation marks and escape characters that are used to make it a valid JSON string.	to_json ('Fred said "Hi."'::text)	"Fred said \"Hi.\""
array_to_json (anyarray [, pretty_bool])	Returns an array as a JSON array. If you enter a multidimensional array, a JSON array of arrays is returned. Note If the value of the pretty_bool parameter is true, line feeds are added between dimension-1 elements.	array_to_json ('{{1,5}, {99,100}}'::int[])	[[1,5],[99,100]]

Function	Description	Example	Result
row_to_json (record [, pretty_bool])	Returns the row as a JSON object. Note If the value of the pretty_bool parameter is true, line feeds are added between dimension-1 elements.	row_to_json (row(1,'foo'))	{"f1":1,"f2":"foo"}

JSON processing functions

The following table describes the functions used to process JSON values.

Function	Return value type	Description	Example	Return result example
json_each(json)	set of key text, value json set of key text, value jsonb	Expands the outermost JSON object into a set of key-value pairs.	select * from json_each('{"a":"foo", "b":"bar"}')	key | value -----+------- a | "foo" b | "bar"
json_each_text(json)	set of key text, value text	Expands the outermost JSON object into a set of key-value pairs. The return values are of the TEXT type.	select * from json_each_text('{"a":"foo", "b":"bar"}')	key | value -----+------- a | foo b | bar
json_extract_path(from_json json, VARIADIC path_elems text[])	json	Returns the JSON value specified by the path_elems parameter. This function is equivalent to the #> operator.	json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}','f4')	{"f5":99,"f6":"foo"}
json_extract_path_text(from_json json, VARIADIC path_elems text[])	text	Returns the JSON value specified by the path_elems parameter as JSON text. This function is equivalent to the #>> operator.	json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}','f4', 'f6')	foo

Function	Return value type	Description	Example	Return result example

`json_object_keys(json)`	`setof text`	Returns a set of keys in the outermost JSON object.	`json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}}')`	`json_object_keys`
				`---`
				`f1`
				`f2`

`json_populate_record(base anyelement, from_json json)`	`anyelement`	Expands the object in the from_json parameter into a row with columns that match the record type defined by the base parameter.	`select * from json_populate_record(null::myrow type, '{"a":1,"b":2}')`	`a | b`
				`---+---`
				`1 | 2`

`json_populate_recordset(base anyelement, from_json json)`	`set of anyelement`	Expands the outermost array of objects in the from_json parameter into a set of rows with columns that match the record type defined by the base parameter.	`select * from json_populate_recordset(null::myrowtype, '[{"a":1,"b":2}, {"a":3,"b":4}]')`	`a | b`
				`---+---`
				`1 | 2`
				`3 | 4`

`json_array_elements(json)`	`set of json`	Expands a JSON array into a set of JSON values.	`select * from json_array_elements('[1,true, [2,false]]')`	`value`
				`---`
				`1`
				`true`
				`[2,false]`

Create a JSON-B function index

You can create GIN and B-tree indexes on JSON-B columns. You can execute the following statements to create a GIN index on a JSON-B column:

```
CREATE INDEX idx_name ON table_name USING gin (idx_col);
CREATE INDEX idx_name ON table_name USING gin (idx_col jsonb_path_ops);
```

> **Note** You can use one of the following operators to create a GIN index on a JSON-B column: the default jsonb_ops operator and the jsonb_path_ops operator. The difference between a jsonb_ops and a jsonb_path_ops GIN index is that the former creates independent index items for each key and value in the data, while the latter creates index items only for each value in the data.

Examples

Create a table.

```
create table tj(id serial, ary int[], obj json, num integer);
=> insert into tj(ary, obj, num) values('{1,5}'::int[], '{"obj":1}', 5);
INSERT 0 1
=> select row_to_json(q) from (select id, ary, obj, num from tj) as q;
row_to_json
--------------------------------------------
{"f1":1,"f2":[1,5],"f3":{"obj":1},"f4":5}
(1 row)
=> insert into tj(ary, obj, num) values('{2,5}'::int[], '{"obj":2}', 5);
INSERT 0 1
=> select row_to_json(q) from (select id, ary, obj, num from tj) as q;
row_to_json
--------------------------------------------
{"f1":1,"f2":[1,5],"f3":{"obj":1},"f4":5}
{"f1":2,"f2":[2,5],"f3":{"obj":2},"f4":5}
(2 rows)
```

> **Note** JSON data cannot be used as partition keys and does not support JSON aggregate functions.

Join multiple tables.

```
create table tj2(id serial, ary int[], obj json, num integer);
=> insert into tj2(ary, obj, num) values('{2,5}'::int[], '{"obj":2}', 5);
INSERT 0 1
=> select * from tj, tj2 where tj.obj->>'obj' = tj2.obj->>'obj';
id | ary | obj | num | id | ary | obj | num
----+-------+-----------+-----+----+-------+-----------+-----
2 | {2,5} | {"obj":2} | 5 | 1 | {2,5} | {"obj":2} | 5
(1 row)
=> select * from tj, tj2 where json_object_field_text(tj.obj, 'obj') = json_object_field_text(tj2.obj, 'obj');
id | ary | obj | num | id | ary | obj | num
----+-------+-----------+-----+----+-------+-----------+-----
2 | {2,5} | {"obj":2} | 5 | 1 | {2,5} | {"obj":2} | 5
(1 row)
```

Create a JSON function index.

```
CREATE TEMP TABLE test_json (
json_type text,
obj json
);
=> insert into test_json values('aa', '{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}');
INSERT 0 1
=> insert into test_json values('cc', '{"f7":{"f3":1},"f8":{"f5":99,"f6":"foo"}}');
INSERT 0 1
=> select obj->'f2' from test_json where json_type = 'aa';
? column?
----------
{"f3":1}
(1 row)
=> create index i on test_json (json_extract_path_text(obj, '{f4}'));
CREATE INDEX
=> select * from test_json where json_extract_path_text(obj, '{f4}') = '{"f5":99,"f6":"foo"}';
json_type | obj
-----------+---------------------------------------------
aa | {"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}
(1 row)
```

Create a JSON-B function index.

```
-- Create a test table and generate data.
CREATE TABLE jtest1 (
```

```
id int,
jdoc json
);

CREATE OR REPLACE FUNCTION random_string(INTEGER)
RETURNS TEXT AS
$BODY$
SELECT array_to_string(
ARRAY (
SELECT substring(
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
FROM (ceil(random()*62))::int FOR 1
)
FROM generate_series(1, $1)
),
''
)
$BODY$
LANGUAGE sql VOLATILE;

insert into jtest1 select t.seq, ('{"a":{"a1":"a1a1", "a2":"a2a2"},
"name":"'||random_string(10)||'","b":"bbbbb"}')::json from
generate_series(1, 10000000) as t(seq);

CREATE TABLE jtest2 (
id int,
jdoc jsonb
);

CREATE TABLE jtest3 (
id int,
jdoc jsonb
);

insert into jtest2 select id, jdoc::jsonb from jtest1;
insert into jtest3 select id, jdoc::jsonb from jtest1;

-- Create an index.
CREATE INDEX idx_jtest2 ON jtest2 USING gin(jdoc);
CREATE INDEX idx_jtest3 ON jtest3 USING gin(jdoc jsonb_path_ops);
```

```
-- Execute a query plan without an index.
EXPLAIN ANALYZE SELECT * FROM jtest1 where jdoc @> '{"name":"N9WP5txmVu"}';
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
------------
Gather Motion 2:1 (slice1; segments: 2) (cost=0.00..162065.73 rows=10100 width=88) (actual time=1343.
248..1777.605 rows=1 loops=1)
-> Seq Scan on jtest2 (cost=0.00..162065.73 rows=5050 width=88) (actual time=0.042..1342.426 rows=1 l
oops=1)
Filter: (jdoc @> '{"name": "N9WP5txmVu"}'::jsonb)
Planning time: 0.172 ms
(slice0) Executor memory: 59K bytes.
(slice1) Executor memory: 91K bytes avg x 2 workers, 91K bytes max (seg0).
Memory used: 2047000kB
Optimizer: Postgres query optimizer
Execution time: 1778.234 ms
(9 rows)

-- Execute a query plan by using the jsonb_ops operator.
EXPLAIN ANALYZE SELECT * FROM jtest2 where jdoc @> '{"name":"N9WP5txmVu"}';
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
------
Gather Motion 2:1 (slice1; segments: 2) (cost=88.27..13517.81 rows=10100 width=88) (actual time=0.655
..0.659 rows=1 loops=1)
-> Bitmap Heap Scan on jtest2 (cost=88.27..13517.81 rows=5050 width=88) (actual time=0.171..0.172 row
s=1 loops=1)
Recheck Cond: (jdoc @> '{"name": "N9WP5txmVu"}'::jsonb)
-> Bitmap Index Scan on idx_jtest2 (cost=0.00..85.75 rows=5050 width=0) (actual time=0.217..0.217 rows
=1 loops=1)
Index Cond: (jdoc @> '{"name": "N9WP5txmVu"}'::jsonb)
Planning time: 0.151 ms
(slice0) Executor memory: 69K bytes.
(slice1) Executor memory: 628K bytes avg x 2 workers, 632K bytes max (seg1). Work_mem: 9K bytes ma
x.
Memory used: 2047000kB
Optimizer: Postgres query optimizer
Execution time: 1.266 ms
(11 rows)

-- Execute the query plan by using the jsonb_path_ops operator.
```

```
-- Execute the query plan by using the jsonb_path_ops operator.
EXPLAIN ANALYZE SELECT * FROM jtest3 where jdoc @> '{"name":"N9WP5txmVu"}';
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
------
Gather Motion 2:1 (slice1; segments: 2) (cost=84.28..13513.81 rows=10101 width=88) (actual time=0.710
..0.711 rows=1 loops=1)
-> Bitmap Heap Scan on jtest3 (cost=84.28..13513.81 rows=5051 width=88) (actual time=0.179..0.181 row
s=1 loops=1)
Recheck Cond: (jdoc @> '{"name": "N9WP5txmVu"}'::jsonb)
-> Bitmap Index Scan on idx_jtest3 (cost=0.00..81.75 rows=5051 width=0) (actual time=0.106..0.106 rows
=1 loops=1)
Index Cond: (jdoc @> '{"name": "N9WP5txmVu"}'::jsonb)
Planning time: 0.144 ms
(slice0) Executor memory: 69K bytes.
(slice1) Executor memory: 305K bytes avg x 2 workers, 309K bytes max (seg1). Work_mem: 9K bytes ma
x.
Memory used: 2047000kB
Optimizer: Postgres query optimizer
Execution time: 1.291 ms
(11 rows)
```

The following example shows how to use Python to access a database:

```
#! /bin/env python
import time
import json
import psycopg2
def gpquery(sql):
conn = None
try:
conn = psycopg2.connect("dbname=sanity1x2")
conn.autocommit = True
cur = conn.cursor()
cur.execute(sql)
return cur.fetchall()
except Exception as e:
if conn:
try:
conn.close()
except:
pass
time.sleep(10)
print e
return None
def main():
sql = "select obj from tj;"
#rows = Connection(host, port, user, pwd, dbname).query(sql)
rows = gpquery(sql)
for row in rows:
print json.loads(row[0])
if __name__ == "__main__":
main()
```

9.Use sort keys in column-oriented tables

> **Notice** Only AnalyticDB for PostgreSQL V4.3 supports sort keys. If you use AnalyticDB for PostgreSQL V6.0 or other versions, ignore the sort key feature.

A sort key is an optional attribute of a column-oriented table. Values in the sort key columns are stored on a disk in sorted order. Sort keys provide the following benefits:

- Speed up and optimize column-oriented tables. The min and max meta information that the system collects seldom overlaps with each other, which offers good filterability.
- Eliminate the need to perform ORDER BY and GROUP BY operations. The data directly read from the disk is ordered as required by the sorting conditions.

This topic describes how to use sort keys in different scenarios.

Define columns as sort keys when you create a table

You can use the `CREATE TABLE` statement to create a table. The following statement shows the syntax of CREATE TABLE:

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ] [column_constraint [ ... ]
[ ENCODING ( storage_directive [,...] ) ]
]
| table_constraint
| LIKE other_table [{INCLUDING | EXCLUDING}
{DEFAULTS | CONSTRAINTS}] ...}
[, ... ] ]
[column_reference_storage_directive [, ] ]
)
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter=value [, ... ] )
[ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
[ TABLESPACE tablespace ]
[ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
[ SORTKEY (column, [ ... ] )]
[ PARTITION BY partition_type (column)
[ SUBPARTITION BY partition_type (column) ]
[ SUBPARTITION TEMPLATE ( template_spec ) ]
[...]
( partition_spec )
| [ SUBPARTITION BY partition_type (column) ]
[...]
( partition_spec
[ ( subpartition_spec
[(...)]
) ]
)
```

Example

```
create table test(date text, time text, open float, high float, low float, volume int) with(APPENDONLY=t
rue,ORIENTATION=column) sortkey (volume);
```

Sort a table

The following statement can be used:

```
VACUUM SORT ONLY [tablename]
```

Modify sort keys

The following statement can be used:

```
ALTER [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name SET SORTKEY (column, [ ... ] )
```

> **Notice** The preceding statement only modifies the catalog and does not sort data. You must execute the `VACUUM SORT ONLY` statement to sort data.

Example

```
alter table test set sortkey (high,low);
```

Precautions

After you perform operations such as INSERT, UPDATE, and DELETE on a table, data in the table is viewed as not sorted by the sort key column. The data read from the disk is not viewed as sorted data in queries. You must execute the `VACUUM SORT ONLY` statement to rearrange the data in the table.

10.Use ANALYZE statements to collect statistics on AnalyticDB for PostgreSQL

Accurate statistics are necessary for the query optimizer to choose an efficient query plan. If the statistics are incorrect or outdated, the query optimizer may generate an inefficient query plan. You can use the ANALYZE statement to update statistics.

The syntax is as follows:

```
ANALYZE [VERBOSE] [ROOTPARTITION [ALL] ] [table [ (column [, ...] ) ]]
```

Generate statistics selectively

The execution of an ANALYZE statement with no parameters updates statistics for all tables in a database. This can be a very long-running process. You can specify a table or column to collect its statistics. If data is changed, you can only execute ANALYZE statements for tables with changed data.

The execution of an ANALYZE statement on a large table takes a long time. Therefore, to shorten the time needed, you can execute only the `ANALYZE table(column, ...)` statement to generate statistics for selected columns, such as columns used in joins, WHERE clauses, SORT clauses, GROUP BY clauses, or HAVING clauses. For a partitioned table, you can find its name in the pg_partitions system catalog by executing the following statement and then execute ANALYZE statements on partitions with changed data:

```
SELECT partitiontablename from pg_partitions WHERE tablename='parent_table';
```

When to execute ANALYZE statements

Execute ANALYZE statements after:

- Data loading
- CREATE INDEX operations
- A large number of INSERT, UPDATE, and DELETE operations

An ANALYZE statement requires only a read lock on a table. Therefore, it can be executed in parallel with other database activities. Do not execute ANALYZE statements when you import data or execute INSERT, UPDATE, DELETE, or CREATE INDEX statements.

11.Use the EXPLAIN statement to read a query plan

A query optimizer uses data statistics maintained by a database to choose the query plan with the lowest possible cost. Cost is measured in disk I/O and is shown as the number of disk page fetches. You can use EXPLAIN and EXPLAIN ANALYZE statements to identify and optimize a query plan. The syntax of an EXPLAIN statement is as follows:

```
EXPLAIN [ANALYZE] [VERBOSE] statement
```

An EXPLAIN statement displays the cost estimated by a query optimizer for a query plan. Example:

```
EXPLAIN SELECT * FROM names WHERE id=22;
```

An EXPLAIN ANALYZE statement enables the statement to be executed, and displays a query plan and additional information. Such information includes the number of executed rows and the runtime. Example:

```
EXPLAIN ANALYZE SELECT * FROM names WHERE id=22;
```

EXPLAIN statement output

A query plan is a tree of nodes. Each node in a query plan represents a single operation such as a table scan, join, aggregation, or sorting. Query plans must be read from the bottom up because each node feeds rows into the node directly above it. The bottom nodes of a query plan are usually table scan operations such as sequential scans, index-based scans, or bitmap index-based scans. If a query requires operations such as join, aggregation, and sorting on rows, there are additional nodes above the scan nodes to perform these operations. The topmost plan nodes are AnalyticDB for PostgreSQL motion nodes: redistribute, broadcast, or gather. These operations move rows between compute nodes during query processing.

The output of an EXPLAIN statement has one line for each node in a plan and shows the node type and the estimates of the execution costs for each plan node:

- cost: It is measured based on the number of disk page fetches. 1.0 equals one sequential disk page read. The first estimate is the startup cost of getting the first row and the second estimate is the total cost of getting all rows.
- rows: indicates the total number of rows generated by a plan node. This number is usually less than the number of rows processed or scanned by the plan node, because of the filter criterion specified in a WHERE clause. The estimate for the topmost node approximates the number of rows that the query actually returns, updates, or deletes.
- width: indicates the total bytes of all the rows that the plan node generates.

Note that:

- The cost of a node includes the cost of all its child nodes. The topmost plan node has the estimated total execution cost for the plan. This is the number the query optimizer intends to minimize.

- The cost only reflects the time taken to execute the query plan in AnalyticDB for PostgreSQL. In particular, the cost does not consider the time taken to transmit result rows to the client.

Example EXPLAIN statement

The following example describes how to read a query plan displayed by an EXPLAIN statement:

```
EXPLAIN SELECT * FROM names WHERE name = 'Joelle';
QUERY PLAN
------------------------------------------------------------
Gather Motion 4:1 (slice1) (cost=0.00..20.88 rows=1 width=13)


-> Seq Scan on 'names' (cost=0.00..20.88 rows=1 width=13)
Filter: name::text ~~ 'Joelle'::text
```

The query optimizer sequentially scans the names table based on the filter criterion specified in the WHERE clause. This means that the query optimizer checks the criterion for each row it scans and only generates rows that meet the criterion. The results of the scan operation are passed up to a gather motion. A gather motion is that compute nodes send rows to the coordinator node. In this example, four compute nodes send rows to the coordinator node, which is indicated by "4:1". The estimated startup cost for this plan is 00.00 (no cost) and the total cost of disk page fetches is 20.88. The query optimizer estimates that this query will return one row.

An EXPLAIN ANALYZE statement displays a query plan and executes statements. The query plan displayed by an EXPLAIN ANALYZE statement shows the actual execution cost along with the estimates provided by the query optimizer. In addition, an EXPLAIN ANALYZE statement displays the following information:

- The total runtime (in milliseconds) during which the query is executed.
- The memory used by each slice of a query plan and the memory reserved for the whole query statement.
- The number of compute nodes involved in a plan node operation. Only compute nodes that return rows are counted.
- The maximum number of rows returned by the compute node that produced the most rows for an operation. If multiple compute nodes produce an equal number of rows, the EXPLAIN ANALYZE statement displays the number of rows generated by the compute node that takes the longest time to produce the rows.
- The ID of the compute node that produces the most rows for an operation.
- The amount of memory required for an operation (work_mem). If the available memory is insufficient to perform an operation, the plan shows the amount of data spilled to disk for the lowest-performing compute node. Example:

  ```
  Work_mem used: 64K bytes avg, 64K bytes max (seg0).
  Work_mem wanted: 90K bytes avg, 90K byes max (seg0) to lessen
  workfile I/O affecting 2 workers.
  ```

- The time (in milliseconds) used by the compute node that produces the most rows to retrieve the first row, and the time taken for that compute node to retrieve all rows.

The following example uses the same query to describe how to read a query plan displayed by an EXPLAIN ANALYZE statement. The bold parts of the plan show actual timing and rows returned for each plan node, along with memory and time statistics for the whole query.

```
EXPLAIN ANALYZE SELECT * FROM names WHERE name = 'Joelle';
QUERY PLAN
------------------------------------------------------------
Gather Motion 2:1 (slice1; segments: 2) (cost=0.00..20.88 rows=1 width=13)
Rows out: 1 rows at destination with 0.305 ms to first row, 0.537 ms to end, start offset by 0.289 ms.
-> Seq Scan on names (cost=0.00..20.88 rows=1 width=13)
Rows out: Avg 1 rows x 2 workers. Max 1 rows (seg0) with 0.255 ms to first row, 0.486 ms to end, start
offset by 0.968 ms.
Filter: name = 'Joelle'::text
Slice statistics:

(slice0) Executor memory: 135K bytes.

(slice1) Executor memory: 151K bytes avg x 2 workers, 151K bytes max (seg0).

Statement statistics: Memory used: 128000K bytes Total runtime: 22.548 ms
```

The total time to run this query is 22.548 milliseconds. The sequential scan operation only has one compute node (seg0) that returns rows, and this compute node only returns one row. It takes 0.255 milliseconds to retrieve the first row and 0.486 milliseconds to scan all rows. The gather motion (compute nodes sending data to the coordinator node) receives one row. The total time for this operation is 0.537 milliseconds.

Common query operators

Scan operators scan rows in a table to find a set of rows. There are the following scan operators:

- Seq Scan: scans all rows in a table.
- Append-only Scan: scans rows in append-optimized row-oriented tables.
- Append-only Columnar Scan: scans rows in append-optimized column-oriented tables.
- Index Scan: traverses a B-tree index to fetch rows from a table.
- Bitmap Append-only Row-oriented Scan: gathers the pointers of rows in an append-optimized table from an index and sorts the pointers by location on a disk.
- Dynamic Table Scan: uses one of the following functions to choose partitions to scan. The Function Scan node contains the function.
  - gp_partition_expansion: selects all partitions in a table.
  - gp_partition_selection: selects a partition based on an equality expression.
  - gp_partition_inversion: selects partitions based on a range expression.

  The Function Scan node passes the list of dynamically selected partitions to the Result node which then passes the list to the Sequence node.

Join operators include:

- Hash Join: builds a hash table from a smaller table with the join column as a hash key, scans a larger table to calculate the hash key for the join column, and probes the hash table to find the rows with the same hash key. Hash joins are typically the fastest joins in AnalyticDB for PostgreSQL. Hash Cond in the query plan identifies the columns that are joined.
- Nested Loop Join: iterates through rows in a larger table and scans the rows in a smaller table on each iteration. This operator requires the broadcast of one table so that all rows in that table can be compared to all rows in the other table. Nested Loop Join performs well for small tables or the tables that are limited by using an index. There are performance implications when Nested Loop Join is used with large tables. Set the enable_nestloop parameter to OFF (default value) to make a query optimizer favor Hash Join.
- Merge Join: sorts two tables and merges them together. This operator is fast for pre-ordered data. To make a query optimizer favor Merge Join, set the enable_mergejoin parameter to ON.

Motion operators move rows between compute nodes. There are the following motion operators:

- Broadcast motion: Each compute node sends its rows to all the other compute nodes so that every compute node has a complete copy of a table. In most cases, a query optimizer only selects a Broadcast motion for small tables. The Broadcast motion is not suitable for large tables. If data is not distributed on the join key, required rows are dynamically redistributed from one of the tables to another compute node.
- Redistribute motion: Each compute node rehashes data and sends its rows to appropriate compute nodes based on the hash key.
- Gather motion: Result data from all compute nodes is assembled and then sent to the coordinator node. This is the final operation for most query plans.

Other operators that occur in query plans include:

- Materialize: materializes a subselect.
- InitPlan: indicates a pre-query. This operator is used in dynamic partition elimination and is executed if the system does not know the values of partitions to be scanned by the query optimizer.
- Sort: sorts rows in preparation for another operation that requires ordered rows, such as Aggregation or Merge Join.
- Group By: groups rows by one or more columns.
- Group/Hash Aggregate: uses a hash algorithm to aggregate rows.
- Append: concatenates data sets when rows scanned from partitions in a partitioned table are combined.
- Filter: selects rows by using the filter criterion specified in a WHERE clause.
- Limit: limits the number of rows returned.

Query optimizer determination

You can view EXPLAIN statement outputs to determine if an ORCA or legacy query optimizer is used to generate a query plan. This information appears at the end of an EXPLAIN statement output. The Settings line displays the setting of the OPTIMIZER parameter. The Optimizer status line displays whether an ORCA or legacy query optimizer generates the query plan.

The following EXPLAIN statement output shows that the query plan is generated by a GPORCA query optimizer.

```
QUERY PLAN
------------------------------------------------------------------------------------
Aggregate (cost=0.00..296.14 rows=1 width=8)
-> Gather Motion 2:1 (slice1; segments: 2) (cost=0.00..295.10 rows=1 width=8)
-> Aggregate (cost=0.00..294.10 rows=1 width=8)
-> Table Scan on part (cost=0.00..97.69 rows=100040 width=1)
Settings: optimizer=on
Optimizer status: PQO version 1.609
(5 rows)
explain select count(*) from part;
```

The following EXPLAIN statement output shows that the query plan is generated by a legacy query optimizer.

```
QUERY PLAN
------------------------------------------------------------------------------------------
Aggregate (cost=3519.05..3519.06 rows=1 width=8)
-> Gather Motion 2:1 (slice1; segments: 2) (cost=3518.99..3519.03 rows=1 width=8)
-> Aggregate (cost=3518.99..3519.00 rows=1 width=8)
-> Seq Scan on part (cost=0.00..3018.79 rows=100040 width=1)
Settings: optimizer=off
Optimizer status: legacy query optimizer
(5 rows)
```

12.Use resource queues for load management

This topic describes how to create and use resource queues in AnalyticDB for PostgreSQL. You can use resource queues to manage and isolate system resources.

Overview

CPU and memory resources for a database instance are limited. These resources have an impact on the query performance of a database. If database loads reach a threshold, all queries compete for the resources. This lowers overall query performance and affects latency-sensitive businesses.

AnalyticDB for PostgreSQL provides resource queues for you to manage system loads. You can specify the following items based on your business requirements: number of concurrent queries that your database can process, memory size available for each query, and number of CPU resources available for each query. This way, the system can offer resources that meet expectations for each query and achieve expected query performance.

Create a resource queue

Execute the following SQL statement to create a resource queue:

```
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])

The queue_attribute parameter specifies the attribute of a resource queue. Valid values:

ACTIVE_STATEMENTS=integer
[ MAX_COST=float [COST_OVERCOMMIT={TRUE|FALSE}] ]
[ MIN_COST=float ]
[ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
[ MEMORY_LIMIT='memory_units' ]

MAX_COST=float [ COST_OVERCOMMIT={TRUE|FALSE} ]
[ ACTIVE_STATEMENTS=integer ]
[ MIN_COST=float ]
[ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
[ MEMORY_LIMIT='memory_units' ]
```

Parameter	Description
ACTIVE_STATEMENTS	The maximum number of active queries in the resource queue at a certain point in time. An active query refers to a query that is being executed.

Parameter	Description
MEMORY_LIMIT	The maximum memory size that can be used by all queries in the resource queue on a single compute node. • The memory size is measured in KB, MB, or GB. • The default value is -1, which indicates an unlimited memory size.
MAX_COST	The maximum cost of a query in the resource queue. The default value is-1, which indicates unlimited cost. Note The cost refers to the cost estimated for a query by the query optimizer in AnalyticDB for PostgreSQL.
COST_OVERCOMMIT	If the MAX_COST parameter is specified, this parameter must be set. • If COST_OVERCOMMIT is set to TRUE, a query whose cost is greater than the value of MAX_COST is executed when the system load is low. • If COST_OVERCOMMIT is set to FALSE, a query whose cost is greater than the value of MAX_COST is rejected.
MIN_COST	The minimum cost of a query in the resource queue. A query whose cost is less than the value of MIN_COST is immediately executed without being queued.
PRIORITY	The priority of the resource queue. Queries in a resource queue with a higher priority are assigned more CPU resources for execution. Valid values: • MIN • LOW • MEDIUM • HIGH • MAX Default value: MEDIUM.

Note When creating a resource queue, you must set either `ACTIVE_STATEMENTS` or `MAX_COST`. If you do not do this, the creation fails.

```
postgres=> CREATE RESOURCE QUEUE adhoc2 WITH (MEMORY_LIMIT='2000MB');
ERROR: at least one threshold ("ACTIVE_STATEMENTS", "MAX_COST") must be specified
```

- Configure the maximum number of active queries

  When you create a resource queue, execute the following statement to configure the maximum number of active queries in this resource queue:

```
CREATE RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=3);
```

In this example, a resource queue named `adhoc` is created, and a maximum of three active queries are allowed for this resource queue at a specific point in time. If there are already three active queries in this resource queue, new queries are queued for execution until the three queries are executed.

- Configure an upper memory limit

  When you create a resource queue, execute the following statement to configure the maximum memory size that can be used by all queries in the resource queue:

  ```
  CREATE RESOURCE QUEUE myqueue WITH (ACTIVE_STATEMENTS=20, MEMORY_LIMIT='2000MB');
  ```

  In this example, a resource queue named `myqueue` is created, a maximum of 20 active queries are allowed for the resource queue at a specific point in time, and all queries in this resource queue can use a maximum of 2,000 MB of memory. Therefore, each query in this resource queue can use a maximum of 100 MB of memory. If a query requires independent memory, you can use the statement_mem parameter to set the memory size for this query. The value of this parameter must be lower than the values of the MEMORY_LIMIT and max_statement_mem parameters. Example:

  ```
  SET statement_mem='1GB';
  SELECT * FROM test_adbpg WHERE col='adb4pg' ORDER BY id;
  RESET statement_mem;
  ```

- Configure the priority of a resource queue

  You can configure priorities for different resource queues to control the use of CPU resources by queries in the resource queues. For example, if AnalyticDB for PostgreSQL receives a large number of concurrent queries, it allocates more resources to queries in a resource queue with a higher priority than queries in a resource queue with a lower priority. This ensures that there are sufficient CPU resources to execute queries in the resource queue with a higher priority.

  You can configure the priority of a resource queue when you create it. Example:

  ```
  CREATE RESOURCE QUEUE executive WITH (ACTIVE_STATEMENTS=3, PRIORITY=MAX);
  ```

  You can also modify the priority of a resource queue after you create it. For more information, see Modify the configuration of a resource queue.

> **Note** The PRIORITY setting for a resource queue differs from the ACTIVE_STATEMENTS and MEMORY_LIMIT settings.
>
> - The values of the ACTIVE_STATEMENTS and MEMORY_LIMIT parameters determine whether a query is admitted to the queue and eventually executed.
> - The PRIORITY setting ensures that after the system starts to execute a query, available CPU resources are dynamically allocated to the query based on the motion status of the system and the priority of the resource queue to which the query belongs.
>
> For example, AnalyticDB for PostgreSQL is executing queries in a resource queue with a lower priority, and a query in a resource queue with a higher priority is admitted and is ready to execute. AnalyticDB for PostgreSQL allocates more CPU resources to the query from the resource queue with the higher priority and reduces the CPU resources for queries in the resource queue with the lower priority. The rules to allocate CPU resources are as follows:
>
> - AnalyticDB for PostgreSQL allocates the same number of CPU resources to queries in resource queues with the same priority.
> - If AnalyticDB for PostgreSQL receives queries in resource queues with high, medium, and low priorities, it allocates 90% of CPU resources to the queries in the high-priority resource queue. Among the remaining 10% of CPU resources, AnalyticDB for PostgreSQL allocates 90% of them to the queries in the medium-priority resource queue and 10% of them to the queries in the low-priority resource queue.

After you create a resource queue, you can execute the `gp_toolkit.gp_resqueue_status` statement to view the status of the resource queue and the resource limits configured for it.

```
postgres=> SELECT * from gp_toolkit.gp_resqueue_status WHERE
postgres-> rsqname='adhoc';
queueid | rsqname | rsqcountlimit | rsqcountvalue | rsqcostlimit | rsqcostvalue | rsqmemorylimit | rsqme
moryvalue | rsqwaiters | rsqholders
---------+---------+---------------+---------------+--------------+--------------+----------------+----------------+--
----------+------------
19283 | adhoc | 3 | 0 | -1 | 0 | -1 | 0 | 0 | 0
(1 row)
```

You cannot create a resource queue within a transaction block. Example:

```
postgres=> begin;
BEGIN
postgres=> CREATE RESOURCE QUEUE test_q WITH (ACTIVE_STATEMENTS=3, PRIORITY=MAX);
ERROR: CREATE RESOURCE QUEUE cannot run inside a transaction block
```

Note Not all SQL statements are restricted by resource queues.

- If the resource_select_only parameter is set to on, SELECT, SELECT INTO, CREATE TABLE AS SELECT, and DECLARE CURSOR statements are restricted by resource queues.
- If the resource_select_only parameter is set to off, INSERT, UPDATE, and DELETE statements are restricted by resource queues in addition to SELECT, SELECT INTO, CREATE TABLE AS SELECT, and DECLARE CURSOR statements.
- In AnalyticDB for PostgreSQL, the resource_select_only parameter is set to off by default.

Assign users to a resource queue

After you create a resource queue, you must assign one or more users to it. Then, AnalyticDB for PostgreSQL can manage resources for queries in the resource queue.

Note

- If a user is not assigned to a resource queue, AnalyticDB for PostgreSQL assigns this user to the pg_default resource queue.
- This resource queue supports a maximum of 500 active queries and has no cost limits. Its priority is medium.

```
postgres=> SELECT * from gp_toolkit.gp_resqueue_status WHERE rsqname='pg_default';
queueid | rsqname | rsqcountlimit | rsqcountvalue | rsqcostlimit | rsqcostvalue | rsqmemorylimit | rs
qmemoryvalue | rsqwaiters | rsqholders
---------+------------+---------------+---------------+--------------+--------------+----------------+----------
-----+------------+------------
6055 | pg_default | 500 | 1 | -1 | 126 | -1 | 2.096128e+09 | 0 | 1
(1 row)
```

Execute either of the following statements to assign a user to a resource queue:

```
ALTER ROLE name RESOURCE QUEUE queue_name;
CREATE ROLE name WITH LOGIN RESOURCE QUEUE queue_name;
```

You can assign a resource queue to a user when the user is created. You can also change the resource queue assigned to a user after the user is created.

Note A user can only belong to one resource queue.

Remove a user from a resource queue

Execute the following statement to remove a user from a resource queue:

```
ALTER ROLE role_name RESOURCE QUEUE none;
```

Modify the configuration of a resource queue

You can use the following statements to modify the configuration of a resource queue.

- Modify the number of active queries:

```
ALTER RESOURCE QUEUE adhoc WITH (ACTIVE_STATEMENTS=5);
```

- Modify the maximum memory size that can be used by all queries and the maximum cost of a query:

```
ALTER RESOURCE QUEUE adhoc WITH (MAX_COST=-1.0, MEMORY_LIMIT='2GB');
```

- Modify the priority:

```
ALTER RESOURCE QUEUE adhoc WITH (PRIORITY=LOW);
ALTER RESOURCE QUEUE reporting WITH (PRIORITY=HIGH);
```

Delete a resource queue

Execute the following statement to delete a resource queue:

```
DROP RESOURCE QUEUE name;
```

> **Note** You are not allowed to delete a resource queue that contains the following items:
>
> - Assigned users
> - Queries in the waiting state

13.Use MetaScan to accelerate queries on column-oriented tables

Note Currently, this feature is only available for AnalyticDB for PostgreSQL V4.3, but not V6.0 nor others.

AnalyticDB for PostgreSQL supports column-oriented storage and features a high data compression ratio and improved query performance. However, it must read data from the entire column or create a B-tree index when processing query criteria with a high filter rate. There are two potential issues with indexing. One is that indexes are not compressed, which results in serious data bloat. The other is that large result sets cause indexes to fail and the cost to be higher than that of a table scan. To address the issues, AnalyticDB for PostgreSQL provides the MetaScan feature, which offers excellent filter performance and occupies minimal storage space.

Enable MetaScan

The MetaScan feature is controlled by the following parameters:

- RDS_ENABLE_CS_ENHANCEMENT

  Specifies whether to collect metadata. If it is set to ON, metadata collection is enabled. If it is set to OFF, metadata collection is disabled. By default, this parameter is set to ON. In this situation, if the data in column-oriented tables changes, the system automatically collects the metadata of the tables. This parameter is an instance-level parameter. If you want to change its setting, contact technical support by submitting a ticket.

- RDS_ENABLE_COLUMN_META_SCAN

  Specifies whether to enable the MetaScan feature for a query. If it is set to ON, the feature is enabled for a query. If it is set to OFF, the feature is disabled for a query. By default, this parameter is set to OFF. This parameter is a session-level parameter. You can view or change its setting by executing the following SQL statements:

```
--- Check whether MetaScan is enabled.
Show RDS_ENABLE_COLUMN_META_SCAN;
--- Disable MetaScan.
Set RDS_ENABLE_COLUMN_META_SCAN = OFF;
--- Enable MetaScan.
Set RDS_ENABLE_COLUMN_META_SCAN = ON;
```

Notice After you set RDS_ENABLE_CS_ENHANCEMENT to OFF, metadata collection is stopped for all column-oriented tables. If you want to use the MetaScan feature after setting RDS_ENABLE_CS_ENHANCEMENT to ON, execute the ALTER TABLE table_name SET WITH (REORGANIZE=TRUE) statement to collect the metadata of the tables again.

Check whether MetaScan is used for a query

You can use the EXPLAIN statement to check whether a SELECT statement uses MetaScan.

```
postgres=# EXPLAIN SELECT * from lineitem where l_shipdate = '2019-01-01 00:00:00';
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Gather Motion 3:1  (slice1; segments: 3)  (cost=0.00..99175.19 rows=2370 width=129)
   -> Append-only Columnar Meta Scan on lineitem  (cost=0.00..99175.19 rows=790 width=129)
         Filter: l_shipdate = '2019-01-01'::date
 Optimizer status: legacy query optimizer
(4 rows)
```

The "Append-only Columnar Meta Scan" node displayed is the MetaScan node. This indicates that the MetaScan feature is used in the query.

Data types and operators supported by MetaScan

MetaScan supports the following data types:

- INT2/INT4/INT8
- FLOAT4/FLOAT8
- TIME/TIMETZ/TIMESTAMP/TIMESTAMPTZ
- VARCHAR/TEXT/BPCHAR
- CASH

MetaScan supports the following operators:

- Equal to (=), less than (<), less than or equal to (≤), greater than (>), and greater than or equal to (≥)
- Logical AND operator

Upgrade existing instances to use MetaScan

If you want to use MetaScan for existing instances, perform the following operations:

- Update the kernel of your AnlayticDB for PostgreSQL instance

  In the AnalyticDB for PostgreSQL console, find the instance whose kernel you want to update and click its ID. In the upper-right corner of the page that appears, click Upgrade Minor Version.

- Update the metadata of tables

  Update the metadata of tables to its latest version after you update the kernel. If RDS_ENABLE_CS_ENHANCEMENT is set to OFF, contact technical support by submitting a ticket to update the metadata. Otherwise, follow these steps:

  i. Use the administrator account to create an update function.

```
CREATE OR REPLACE FUNCTION UPGRADE_AOCS_TABLE_META(tname TEXT) RETURNS BOOL AS $$
DECLARE
tcount INT := 0;
BEGIN
-- CHECK TABLE NAME
EXECUTE 'SELECT COUNT(1) FROM PG_AOCSMETA WHERE RELID = ''' || tname || '''::REGCLASS' INTO
tcount;
IF tcount IS NOT NULL THEN
IF tcount > 1 THEN
RAISE EXCEPTION 'found more than one table of name %', tname;
ELSEIF tcount = 0 THEN
RAISE EXCEPTION 'not found target table in pg_aocsmeta, table name:%', tname;
END IF;
END IF;

EXECUTE 'ALTER TABLE ' || tname || ' SET WITH(REORGANIZE=TRUE)';
RETURN TRUE;
END;
$$ LANGUAGE PLPGSQL;
```

ii. Execute the following SQL statement to update the target table as the administrator or table owner:

```
SELECT UPGRADE_AOCS_TABLE_META(table_name);
```

iii. Execute the following SQL statement to check the metadata version:

```
SELECT version = 4 AS is_latest_version FROM pg_appendonly WHERE relid = 'test'::REGCLASS
```

Enhance MetaScan performance by using SortKey

SortKey is a feature of AnlayticDB for PostgreSQL. It sorts data in tables by a specific column. Using MetaScan together with SortKey improves the performance of MetaScan. Column-oriented tables store data to blocks. MetaScan uses metadata to check whether blocks meet conditions. It skips blocks that do not meet conditions. This reduces I/O and improves scanning performance. If data in filtered columns is distributed in different blocks, all blocks need to be scanned even though most required data is filtered. If you create a SortKey for each filtered column, the same data in a column is combined into several consecutive blocks. This way, MetaScan can filter out blocks that do not meet conditions to improve scanning performance.

For more information about how to create a SortKey, see Use sort keys in column-oriented tables.

Limits

MetaScan is incompatible with the ORCA optimizer and is unavailable if the ORCA optimizer is used. You can execute the following statement to check whether the ORCA optimizer is used:

```
SHOW OPTIMIZER;
```

If ON is returned, the ORCA optimizer is used.