ALIBABA CLOUD

Alibaba Cloud

AnalyticDB for PostgreSQL

Beginner Developer Guide

Document Version: 20220401

Alibaba Cloud

Legal disclaimer

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 922337203 6854775807	An integer within a large range.
bigserial	serial8	8 bytes	1 to 922337203 6854775807	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.
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.

Data type	Alias	Length	Range	Description
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 float	8 bytes	15 digits	Variable precision, which is inaccurate.
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.
macaddr		6 bytes		A MAC address.
money		8 bytes	-92233720368547758.08 to +92233720368547758.07	An amount of money.

Data type	Alias	Length	Range	Description
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.
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.

Data type	Alias	Length	Range	Description
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 statements

This topic describes the SQL statements that are available in and the syntax of the statements.

- ABORT
- ALTER AGGREGATE
- ALTER CONVERSION
- ALTER DATABASE
- ALTER DOMAIN
- ALTER EXTERNAL TABLE
- ALTER FUNCTION
- ALTER GROUP
- ALTER INDEX
- ALTER OPERATOR
- ALTER RESOURCE QUEUE
- ALTER ROLE
- ALTER SCHEMA
- ALTER SEQUENCE
- ALTER TABLE
- ALTER TYPE
- ALTER USER
- ANALYZE
- BEGIN
- CHECKPOINT
- CLOSE
- CLUSTER
- COMMENT
- COMMIT
- COPY
- CREATE AGGREGATE
- CREATE CAST
- CREATE CONVERSION
- CREATE DATABASE
- CREATE DOMAIN
- CREATE EXTENSION
- CREATE EXTERNAL TABLE
- CREATE FUNCTION
- CREATE GROUP
- CREATE INDEX
- CREATE LIBRARY
- 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, see 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, see 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, see 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, see 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 constr
aint_name [RESTRICT | CASCADE] ALTER DOMAIN name OWNER TO new_owner ALTER DOMAIN name SET S
CHEMA new_schema
```

For more information, see 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 RE
NAME TO new_name ALTER EXTERNAL TABLE name SET SCHEMA new_schema ALTER EXTERNAL TABLE name
action [, ... ]
```

For more information, see 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 FU
NCTION name ( [ [argmode] [argname] argtype [, ...] ] ) OWNER TO new_owner ALTER FUNCTION n
ame ( [ [argmode] [argname] argtype [, ...] ] ) SET SCHEMA new_schema
```

For more information, see 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, see 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 I
NDEX name SET ( FILLFACTOR = value ) ALTER INDEX name RESET ( FILLFACTOR )
```

For more information, see ALTER INDEX.

ALTER OPERATOR

Changes the definition of an operator.

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

For more information, see ALTER OPERATOR.

ALTER RESOURCE QUEUE

Changes the limits of a resource queue.

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

For more information, see ALTER RESOURCE QUEUE.

ALTER ROLE

Changes a database role (user or group).

```
ALTER ROLE name RENAME TO newname ALTER ROLE name SET config_parameter {TO | =} {value | DE
FAULT} ALTER ROLE name RESET config_parameter ALTER ROLE name RESOURCE QUEUE {queue_name |
NONE} ALTER ROLE name [ [WITH] option [ ... ] ]
```

For more information, see 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, see ALTER SCHEMA.

ALTER SEQUENCE

Changes the definition of a sequence generator.

```
ALTER SEQUENCE name [INCREMENT [ BY ] increment] [MINVALUE minvalue | NO MINVALUE] [MAXVALU
E 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, see 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 (co
lumn, [ ... ] ) | DISTRIBUTED RANDOMLY | WITH (REORGANIZE=true|false) ALTER TABLE [ONLY] na
me action [, ... ] ALTER TABLE name [ ALTER PARTITION { partition_name | FOR (RANK(number))
| FOR (value) } partition_action [...] ] partition_action
```

For more information, see 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, see 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 | DE
FAULT} ALTER USER name RESET config_parameter ALTER USER name [ [WITH] option [ ... ] ]
```

For more information, see ALTER USER.

ANALYZE

Collects statistics about a database.

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

For more information, see ANALYZE.

BEGIN

Starts a transaction block.

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

For more information, see BEGIN.

CHECKPOINT

Forces a transaction log checkpoint.

```
CHECKPOINT
```

For more information, see CHECKPOINT.

CLOSE

Closes a cursor.

```
CLOSE cursor_name
```

For more information, see 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, see CLUSTER.

COMMENT

Defines or changes the comments of an object.

```
COMMENT ON { TABLE object_name | COLUMN table_name.column_name | AGGREGATE agg_name (agg_ty
pe [, ...]) | 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 OB
JECT 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 obje
ct_name | TABLESPACE object_name | TRIGGER trigger_name ON table_name | TYPE object_name |
VIEW object_name } IS 'text'
```

For more information, see COMMENT.

COMMIT

Commits the current transaction.

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

For more information, see COMMIT.

COPY

Copies data between a file and a table.

```
COPY table [(column [, ...])] FROM {'file' | STDIN} [ [WITH] [BINARY] [OIDS] [HEADER] [DELI
MITER [ AS ] 'delimiter'] [NULL [ AS ] 'null string'] [ESCAPE [ AS ] 'escape' | 'OFF'] [NEW
LINE [ 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] [OI
DS] [HEADER] [DELIMITER [ AS ] 'delimiter'] [NULL [ AS ] 'null string'] [ESCAPE [ AS ] 'esc
ape' | 'OFF'] [CSV [QUOTE [ AS ] 'quote'] [FORCE QUOTE column [, ...]] ] [IGNORE EXTERNAL P
ARTITIONS ]
```

For more information, see COPY.

CREATE AGGREGATE

Defines a new aggregate function.

```
CREATE [ORDERED] AGGREGATE name (input_data_type [ , ... ]) ( SFUNC = sfunc, STYPE = state_
data_type [, PREFUNC = prefunc] [, FINALFUNC = ffunc] [, INITCOND = initial_condition] [, S
ORTOP = sort_operator] )
```

For more information, see CREATE AGGREGATE.

CREATE CAST

Defines a new cast.

```
CREATE CAST (sourcetype AS targettype) WITH FUNCTION funcname (argtypes) [AS ASSIGNMENT | A
S IMPLICIT] CREATE CAST (sourcetype AS targettype) WITHOUT FUNCTION [AS ASSIGNMENT | AS IMP
LICIT]
```

For more information, see CREATE CAST.

CREATE CONVERSION

Defines a new encoding conversion.

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

For more information, see CREATE CONVERSION.

CREATE DATABASE

Creates a new database.

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

For more information, see CREATE DATABASE.

CREATE DOMAIN

Defines a new domain.

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

For more information, see CREATE DOMAIN.

CREATE EXTENSION

Registers 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, see CREATE EXTENSION.

CREATE EXTERNAL TABLE

Defines 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] 'delim
iter'] [NULL [AS] 'null string'] [FORCE NOT NULL column [, ...]] [ESCAPE [AS] 'escape'] [NE
WLINE [ AS ] 'LF' | 'CR' | 'CRLF'] [FILL MISSING FIELDS] )] [ ENCODING 'encoding' ] [ [LOG
ERRORS [INTO error_table]] SEGMENT REJECT LIMIT count [ROWS | PERCENT] ] CREATE WRITABLE EX
TERNAL TABLE table_name ( column_name data_type [, ...] | LIKE other_table ) LOCATION ('oss
protocol') 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 'en
coding' ] [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ] ossprotocol: oss://o
ss_endpoint prefix=prefix_name id=userossid key=userosskey bucket=ossbucket compressiontype
=[none|gzip] async=[true|false] ossprotocol: oss://oss_endpoint dir=[folder/[folder/]...]/f
ile_name id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[tr
ue|false] ossprotocol: oss://oss_endpoint filepath=[folder/[folder/]...]/file_name id=usero
ssid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|false]
```

For more information, see CREATE EXTERNAL TABLE.

CREATE FUNCTION

Defines 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 D
EFINE | COST execution_cost | SET configuration_parameter { TO value | = value | FROM CURRE
NT } | AS 'definition' | AS 'obj_file', 'link_symbol' } ... [ WITH ({ DESCRIBE = describe_f
unction } [, ...] ) ]
```

For more information, see CREATE FUNCTION.

CREATE GROUP

Defines a new database role.

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

For more information, see CREATE GROUP.

CREATE INDEX

Defines a new index.

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

For more information, see CREATE INDEX.

CREATE LIBRARY

Creates a custom software table.

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

For more information, see CREATE LIBRARY.

CREATE OPERATOR

Defines a new 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_tha
n_op] [, GTCMP = greater_than_op] )
```

For more information, see CREATE OPERATOR.

CREATE RESOURCE QUEUE

Defines a new resource queue.

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

For more information, see CREATE RESOURCE QUEUE.

CREATE ROLE

Defines a new database role (user or group).

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

For more information, see CREATE ROLE.

CREATE RULE

Defines a new rewrite rule.

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

For more information, see CREATE RULE.

CREATE SCHEMA

Defines a new schema.

```
CREATE SCHEMA schema_name [AUTHORIZATION username] [schema_element [ ... ]] CREATE SCHEMA A
UTHORIZATION rolename [schema_element [ ... ]]
```

For more information, see CREATE SCHEMA.

CREATE SEQUENCE

Defines a new sequence generator.

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

For more information, see CREATE SEQUENCE.

CREATE TABLE

Defines a new 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 par
tition_type (column) ] [ SUBPARTITION TEMPLATE ( template_spec ) ] [...] ( partition_spec )
| [ SUBPARTITION BY partition_type (column) ] [...] ( partition_spec [ ( subpartition_spec
[(...)] ) ] )
```

Note AnalyticDB for PostgreSQL instances in Serverless mode do not support the WITH clause. The system automatically selects the optimal algorithm based on the data type.

For more information, see CREATE TABLE.

CREATE TABLE AS

Defines a new 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 RANDOML
Y]
```

For more information, see CREATE TABLE AS.

CREATE TRIGGER

Defines a new trigger.

```
CREATE TRIGGER name {BEFORE | AFTER} {event [OR ...]}        ON table [ FOR [EACH] {ROW | S
TATEMENT} ]         EXECUTE PROCEDURE funcname ( arguments )
```

For more information, see CREATE TRIGGER.

CREATE TYPE

Defines a new 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 | VARIA
BLE}] [, PASSEDBYVALUE] [, ALIGNMENT = alignment] [, STORAGE = storage] [, DEFAULT = defaul
t] [, ELEMENT = element] [, DELIMITER = delimiter] ) CREATE TYPE name
```

For more information, see CREATE TYPE.

CREATE USER

Defines a database role that has the LOGIN permission.

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

For more information, see CREATE USER.

CREATE VIEW

Defines a new view.

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

For more information, see CREATE VIEW.

DEALLOCATE

Deallocates a prepared statement.

```
DEALLOCATE [PREPARE] name
```

For more information, see DEALLOCATE.

DECLARE

Defines a cursor.

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

For more information, see 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, see DELETE.

DROP AGGREGATE

Deletes an aggregate function.

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

For more information, see DROP AGGREGATE.

DROP CAST

Deletes a cast.

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

For more information, see DROP CAST.

DROP CONVERSION

Deletes a conversion.

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

For more information, see DROP CONVERSION.

DROP DATABASE

Deletes a database.

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

For more information, see DROP DATABASE.

DROP DOMAIN

Deletes a domain.

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

For more information, see DROP DOMAIN.

DROP EXTENSION

Deletes an extension from a database.

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

For more information, see DROP EXTENSION.

DROP EXTERNAL TABLE

Deletes the definition of an external table.

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

For more information, see DROP EXTERNAL TABLE.

DROP FUNCTION

Deletes a function.

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

For more information, see DROP FUNCTION.

DROP GROUP

Deletes a database role.

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

For more information, see DROP GROUP.

DROP INDEX

Deletes an index.

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

For more information, see DROP INDEX.

DROP LIBRARY

Deletes a custom library package.

```
DROP LIBRARY library_name
```

For more information, see DROP LIBRARY.

DROP OPERATOR

Deletes an operator.

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

For more information, see DROP OPERATOR.

DROP OWNED

Deletes database objects owned by a database role.

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

For more information, see DROP OWNED.

DROP RESOURCE QUEUE

Deletes a resource queue.

```
DROP RESOURCE QUEUE queue_name
```

For more information, see DROP RESOURCE QUEUE.

DROP ROLE

Deletes a database role.

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

For more information, see DROP ROLE.

DROP RULE

Deletes a rewrite rule.

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

For more information, see DROP RULE.

DROP SCHEMA

Deletes a schema.

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

For more information, see DROP SCHEMA.

DROP SEQUENCE

Deletes a sequence.

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

For more information, see DROP SEQUENCE.

DROP TABLE

Deletes a table.

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

For more information, see DROP TABLE.

DROP TYPE

Deletes a data type.

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

For more information, see DROP TYPE.

DROP USER

Deletes a database role.

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

For more information, see DROP USER.

DROP VIEW

Deletes a view.

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

For more information, see DROP VIEW.

END

Commits the current transaction.

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

For more information, see END.

EXECUTE

Executes a prepared SQL statement.

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

For more information, see EXECUTE.

EXPLAIN

Shows the query plan of a statement.

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

For more information, see EXPLAIN.

FETCH

Retrieves rows from a query by using a cursor.

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

For more information, see FETCH.

GRANT

Defines access permissions.

```
GRANT { {SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER | TRUNCATE } [,...] | ALL
[PRIVILEGES] } ON [TABLE] tablename [, ...] TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTI
ON] GRANT { {USAGE | SELECT | UPDATE} [,...] | ALL [PRIVILEGES] } ON SEQUENCE sequencename
[, ...] TO { rolename | PUBLIC } [, ...] [WITH GRANT OPTION] GRANT { {CREATE | CONNECT | TE
MPORARY | TEMP} [,...] | ALL [PRIVILEGES] } ON DATABASE dbname [, ...] TO {rolename | PUBLI
C} [, ...] [WITH GRANT OPTION] GRANT { EXECUTE | ALL [PRIVILEGES] } ON FUNCTION funcname (
[ [argmode] [argname] argtype [, ...] ] ) [, ...] TO {rolename | PUBLIC} [, ...] [WITH GRAN
T OPTION] GRANT { USAGE | ALL [PRIVILEGES] } ON LANGUAGE langname [, ...] TO {rolename | PU
BLIC} [, ...] [WITH GRANT OPTION] GRANT { {CREATE | USAGE} [,...] | ALL [PRIVILEGES] } ON S
CHEMA schemaname [, ...] TO {rolename | PUBLIC} [, ...] [WITH GRANT OPTION] GRANT { CREATE
| ALL [PRIVILEGES] } ON TABLESPACE tablespacename [, ...] TO {rolename | PUBLIC} [, ...] [W
ITH GRANT OPTION] GRANT parent_role [, ...] TO member_role [, ...] [WITH ADMIN OPTION] GRAN
T { SELECT | INSERT | ALL [PRIVILEGES] } ON PROTOCOL protocolname TO username
```

For more information, see GRANT.

INSERT

Creates new rows in a table.

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

For more information, see INSERT.

LOAD

Loads or reloads a shared library file.

```
LOAD 'filename'
```

For more information, see LOAD.

LOCK

Locks a table.

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

For more information, see LOCK.

MOVE

Positions a cursor.

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

For more information, see MOVE.

PREPARE

Prepares a statement for execution.

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

For more information, see 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, see REASSIGN OWNED.

REINDEX

Rebuilds an index.

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

For more information, see REINDEX.

RELEASE SAVEPOINT

Destroys a defined savepoint.

```
RELEASE [SAVEPOINT] savepoint_name
```

For more information, see RELEASE SAVEPOINT.

RESET

Restores the value of a system configuration parameter to the default value.

```
RESET configuration_parameter RESET ALL
```

For more information, see RESET.

REVOKE

Revokes access permissions.

```
REVOKE [GRANT OPTION FOR] { {SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER | TRU
NCATE } [,...] | 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 } [, ...] [CAS
CADE | RESTRICT] REVOKE [GRANT OPTION FOR] { {CREATE | CONNECT | TEMPORARY | TEMP} [,...] |
ALL [PRIVILEGES] } ON DATABASE dbname [, ...] FROM {rolename | PUBLIC} [, ...] [CASCADE | R
ESTRICT] REVOKE [GRANT OPTION FOR] {EXECUTE | ALL [PRIVILEGES]} ON FUNCTION funcname ( [[ar
gmode] [argname] argtype [, ...]] ) [, ...] FROM {rolename | PUBLIC} [, ...] [CASCADE | RES
TRICT] REVOKE [GRANT OPTION FOR] {USAGE | ALL [PRIVILEGES]} ON LANGUAGE langname [, ...] FR
OM {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 TABLES
PACE tablespacename [, ...] FROM { rolename | PUBLIC } [, ...] [CASCADE | RESTRICT] REVOKE
[ADMIN OPTION FOR] parent_role [, ...] FROM member_role [, ...] [CASCADE | RESTRICT]
```

For more information, see REVOKE.

ROLLBACK

Aborts the current transaction.

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

For more information, see ROLLBACK.

ROLLBACK TO SAVEPOINT

Rolls back the current transaction to a savepoint.

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

For more information, see ROLLBACK TO SAVEPOINT.

SAVEPOINT

Defines a new savepoint within the current transaction.

```
SAVEPOINT savepoint_name
```

For more information, see SAVEPOINT.

SELECT

Retrieves rows from a table or view.

```
[ WITH with_query [, ...] ] SELECT [ALL | DISTINCT [ON (expression [, ...])]] * | expressio
n [[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 operat
or] [NULLS {FIRST | LAST}] [, ...]] [LIMIT {count | ALL}] [OFFSET start] [FOR {UPDATE | SHA
RE} [OF table_name [, ...]] [NOWAIT] [...]]
```

For more information, see SELECT.

SELECT INTO

Defines a new table from the results of a query.

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

For more information, see SELECT INTO.

SET

Changes the value of a database configuration parameter.

```
SET [SESSION | LOCAL] configuration_parameter {TO | =} value | 'value' | DEFAULT} SET [SESS
ION | LOCAL] TIME ZONE {timezone | LOCAL | DEFAULT}
```

For more information, see SET.

SET ROLE

Sets the current role identifier of the current session.

```
SET [SESSION | LOCAL] ROLE rolename SET [SESSION | LOCAL] ROLE NONE RESET ROLE
```

For more information, see SET ROLE.

SET SESSION AUTHORIZATION

Sets the session role identifier and the current role identifier of the current session.

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION rolename SET [SESSION | LOCAL] SESSION AUTHORIZ
ATION DEFAULT RESET SESSION AUTHORIZATION
```

For more information, see SET SESSION AUTHORIZATION.

SET TRANSACTION

Sets 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, see SET TRANSACTION.

SHOW

Shows the value of a system configuration parameter.

```
SHOW configuration_parameter SHOW ALL
```

For more information, see SHOW.

START TRANSACTION

Starts a transaction block.

```
START TRANSACTION [SERIALIZABLE | READ COMMITTED | READ UNCOMMITTED] [READ WRITE | READ ONL
Y]
```

For more information, see START TRANSACTION.

TRUNCATE

Clears all rows of a table.

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

For more information, see TRUNCATE.

UPDATE

Updates 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, see UPDATE.

VACUUM

Collects garbage and optionally analyzes a database.

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

For more information, see VACUUM.

VALUES

Computes a set of rows.

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

For more information, see 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_supe
ruser ...
```

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; REVO
KE DELETE ON TABLE t1 FROM normal_user1;
```

References

For more information, visit Managing Roles and Privileges.

4.Use DML to insert, update, and delete data

This topic describes how to use DML to insert, update, and delete data in .

Insert data

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

You can execute the `INSERT` statement to insert 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', 1.9
9), (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';
```

Update data

You can execute the `UPDATE` statement to update 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.

Constraints

- The column that is defined as the distribution key cannot be updated.
- The column that is 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 the rows in which the value in the price column is 5 to 10:

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

Delete data

You can execute the `DELETE` statement to delete 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.

Constraints

- 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

You can execute the `TRUNCATE` statement to quickly remove all rows from a table. The truncate operation does not truncate tables that are inherited from the table and the rewrite rules of the ON DELETE clause because the operation does not scan the table and only truncates rows in the table. Syntax:

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

For more information, visit TRUNCATE.

Example

Execute the following statement to remove all rows from a table named mytable:

```
TRUNCATE mytable;
```

5.Use INSERT ON CONFLICT to overwrite data

This topic describes how to use INSERT ON CONFLICT 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 V6.0 and not supported in AnalyticDB for PostgreSQL V4.3.

Constraints

- Only AnalyticDB for PostgreSQL V6.0 supports the overwrite feature. AnalyticDB for PostgreSQL V4.3 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 can be a partitioned table only when the minor version of the instance is 6.3.6.1 or later. For more information about how to upgrade the minor version, see Upgrade the engine version.
- Distribution columns and primary key columns cannot be updated in the UPDATE SET clause.
- Subqueries cannot be executed in the UPDATE WHERE clause.
- The table cannot be an updatable view.
- Multiple data records for a primary key cannot be inserted in an INSERT statement. This is a universal limit based on the standard SQL syntax.

Statement

The overwrite syntax is based on the following INSERT statement:

```
[ 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 ON CONFLICT clause can be added to overwrite data. The clause consists of the conflict_target and conflict_action parameters. The following table describes the parameters.

Parameter	Description
conflict_target	• If conflict_action is set to Do Update, you must use conflict_target to specify the primary key or unique index column that is used to define a conflict. • If conflick_action is set to Do Nothing, you can omit conflict_target.
conflict_action	Specify the action to execute after a conflict. Valid values: • DO NOTHING: indicates that the data to insert is discarded if a data conflict occurs in columns specified by conflict_target. • DO UPDATE: indicates that the data is overwritten based on the following UPDATE clause if a data conflict occurs in columns specified by conflict_target.

Examples

Execute the following statement to create a table named t1. Set four columns in the table and specify a as the primary key:

```
CREATE TABLE t1 (a int PRIMARY KEY, b int, c int, d int DEFAULT 0);
```

Execute the following statement to insert a row of data in which the value of the a primary key is 0:

```
INSERT INTO t1 VALUES (0,0,0,0);
```

Execute the following statement to query the t1 table:

```
SELECT * FROM t1;
```

The following result is returned:

```
 a | b | c | d
---+---+---+---
 0 | 0 | 0 | 0
(1 row)
```

If the following statement is executed to insert another row of data into the t1 table, and the inserted value of the a primary key is 0, an error is returned:

```
INSERT INTO t1 VALUES (0,1,1,1);
```

A similar error message is returned:

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

To prevent the preceding error message, you can use INSERT ON CONFLICT described in this topic.

- Add the following ON CONFLICT DO NOTHING clause to ignore the insertion of the conflicting data. This is applicable to scenarios where conflicting data is discarded.

  The following statement provides an example on this scenario:

  ```
  INSERT INTO t1 VALUES (0,1,1,1) ON CONFLICT DO NOTHING;
  ```

  Execute the following statement to query the t1 table:

  ```
  SELECT * FROM t1;
  ```

  The following result is returned. No operation is performed on the t1 table.

  ```
   a | b | c | d
  ---+---+---+---
   0 | 0 | 0 | 0
  (1 row)
  ```

- Add the following ON CONFLICT DO UPDATE clause to update non-primary key columns. This is applicable to scenarios where all columns in a table are overwritten.

  The following statement provides an example on this scenario:

  ```
  INSERT INTO t1 VALUES (0,2,2,2) ON CONFLICT (a) DO UPDATE SET (b, c, d) = (excluded.b, ex
  cluded.c, excluded.d);
  ```

  or

  ```
  INSERT INTO t1 VALUES (0,2,2,2) ON CONFLICT (a) DO UPDATE SET b = excluded.b, c = exclude
  d.c, d = excluded.d;
  ```

  In the DO UPDATE SET clause, you can use excluded to represent the pseudo-table composed of conflicting data. In the case of a primary key conflict, the column values in the pseudo-table are referenced to overwrite the column values in the t1 table. In the preceding statement, the inserted data `(0,2,2,2)` constitutes a pseudo-table named excluded that contains a row and four columns. You can use `excluded.b, excluded.c, excluded.d` to reference the columns in the pseudo-table.

  Execute the following statement to query the t1 table:

  ```
  SELECT * FROM t1;
  ```

  The following result is returned. The non-primary key columns in the t1 table are updated.

  ```
   a | b | c | d
  ---+---+---+---
   0 | 2 | 2 | 2
  (1 row)
  ```

In addition to the preceding scenarios, you can use INSERT ON CONFLICT in the following scenarios:

- Use INSERT ON CONFLICT to overwrite data in some columns when a primary key conflict occurs. This is applicable to scenarios where some columns are overwritten based on conflicting data.

  For example, execute the following statement to overwrite only data in the c column when a primary key conflict occurs:

```
INSERT INTO t1 VALUES (0,0,3,0) ON CONFLICT (a) DO UPDATE SET c = excluded.c;
```

Execute the following statement to query the t1 table:

```
SELECT * FROM t1;
```

The following result is returned:

```
 a | b | c | d
---+---+---+---
 0 | 2 | 3 | 2
(1 row)
```

- Execute the following INSERT ON CONFLICT statement to update data in some columns. This is applicable to scenarios where some columns are updated based on original data.

  For example, execute the following statement to add 1 to data in the d column when a primary key conflict occurs:

  ```
  INSERT INTO t1 VALUES (0,0,3,0) ON CONFLICT (a) DO UPDATE SET d = t1.d + 1;
  ```

  Execute the following statement to query the t1 table:

  ```
  SELECT * FROM t1;
  ```

  The following result is returned:

  ```
   a | b | c | d
  ---+---+---+---
   0 | 2 | 3 | 3
  (1 row)
  ```

- Execute the following INSERT ON CONFLICT statement to set some column values to default values:

  For example, execute the following statement to set values in the d column to default values when the primary key conflict occurs. The default value of the d column is 0.

  ```
  INSERT INTO t1 VALUES (0,0,3,0) ON CONFLICT (a) DO UPDATE SET d = default;
  ```

  Execute the following statement to query the t1 table:

  ```
  SELECT * FROM t1;
  ```

  The following result is returned:

  ```
   a | b | c | d
  ---+---+---+---
   0 | 2 | 3 | 0
  (1 row)
  ```

- Execute the following statements to insert multiple rows to the t1 table.

- For example, execute the following statement to insert two rows of data. For the row that conflicts with the primary key, no operation is performed. For the row that does not conflict with the primary key, the row is inserted as expected.

  ```
  INSERT INTO t1 VALUES (0,0,0,0), (1,1,1,1) ON CONFLICT DO NOTHING;
  ```

  Execute the following statement to query the t1 table:

  ```
  SELECT * FROM t1;
  ```

  The following result is returned:

  ```
   a | b | c | d
  ---+---+---+---
   0 | 2 | 3 | 0
   1 | 1 | 1 | 1
  (2 rows)
  ```

- For example, execute the following statement to insert two rows of data. For the row which conflicts with the primary key, the data is overwritten. For the row which does not conflict with the primary key, the row is inserted as expected.

  ```
  INSERT INTO t1 VALUES (0,0,0,0), (2,2,2,2) ON CONFLICT (a) DO UPDATE SET (b, c, d) = (e
  xcluded.b, excluded.c, excluded.d);
  ```

  Execute the following statement to query the t1 table:

  ```
  SELECT * FROM t1;
  ```

  The following result is returned:

  ```
   a | b | c | d
  ---+---+---+---
   0 | 0 | 0 | 0
   1 | 1 | 1 | 1
   2 | 2 | 2 | 2
  (3 rows)
  ```

- Execute the INSERT ON CONFLICT statement to insert data obtained from subqueries to overwrite data when a primary key conflict occurs. This is applicable to merging data from two tables or performing complex INSERT INTO and SELECT statements.

  Execute the following statement to create a table named t2 that has the same schema as the t1 table:

  ```
  CREATE TABLE t2 (like t1);
  ```

  Execute the following statement to insert two rows of data to the t2 table:

  ```
  INSERT INTO t2 VALUES (2,22,22,22),(3,33,33,33);
  ```

  Execute the following statement to insert data from the t2 table into the t1 table. If a primary key conflict occurs, overwrite non-primary key columns.

```
INSERT INTO t1 SELECT * FROM t2 ON CONFLICT (a) DO UPDATE SET (b, c, d) = (excluded.b, ex
cluded.c, excluded.d);
```

Execute the following statement to query the t1 table:

```
SELECT * FROM t1;
```

The following result is returned:

```
 a | b  | c  | d
---+----+----+----
 0 |  0 |  0 |  0
 1 |  1 |  1 |  1
 2 | 22 | 22 | 22
 3 | 33 | 33 | 33
(4 rows)
```

6.Use COPY ON CONFLICT to overwrite data

COPY ON CONFLICT is a newly supported statement in AnalyticDB for PostgreSQL to overwrite data. When you execute COPY ON CONFLICT statements, you can check constraints only for an entire table and overwrite data only of an entire column.

AnalyticDB for PostgreSQL allows you to import data by executing a COPY statement. However, if the data that you want to import conflicts with table constraints, the COPY statement may fail with an error message. To resolve this problem, AnalyticDB for PostgreSQL provides the COPY ON CONFLICT statement to overwrite data or ignore the write operation.

Note

The COPY ON CONFLICT statement is supported only by AnalyticDB for PostgreSQL V6.0 instances in minor version 20210528 or later. To use this statement, we recommend that you update your instance to the latest minor version. For more information about how to update the minor version, see Upgrade the engine version.

Constraints

- The table to which you want to import data must be a heap table, but not an append-optimized (AO) table. This is because AO tables do not support unique indexes.
- The table can be a partitioned table only when the minor version of the instance is 6.3.6.1 or later. For more information about how to update the minor version, see Upgrade the engine version.
- The table cannot be an updatable view.
- COPY ON CONFLICT can include only the COPY FROM clause and cannot include the COPY TO clause.
- COPY ON CONFLICT does not allow you to specify the CONFLICT index parameter. By default, COPY ON CONFLICT determines all the constrained columns. If the CONFLICT index parameter is specified, the COPY statement fails with an error message.

```
COPY NATION FROM stdin DO ON CONFLICT(n_nationkey) DO UPDATE;
ERROR:  COPY ON CONFLICT does NOT support CONFLICT index params
```

- COPY ON CONFLICT does not allow you to specify the UPDATE SET parameter. By default, COPY ON CONFLICT updates all columns. If the UPDATE SET parameter is specified, the COPY statement fails with an error message. Example:

```
COPY NATION FROM stdin DO ON CONFLICT DO UPDATE SET n_nationkey = excluded.n_nationkey;
ERROR:  COPY ON CONFLICT does NOT support UPDATE SET targets
```

Syntax

```
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] ]
    [DO ON CONFLICT DO UPDATE | NOTHING]
```

COPY ON CONFLICT supports the DO ON CONFLICT DO UPDATE and DO ON CONFLICT DO NOTHING clauses.

- The DO ON CONFLICT DO UPDATE clause is used to update an entire column when data conflicts with table constraints.
- The DO ON CONFLICT DO NOTHING clause is used to ignore insertion when data conflicts with table constraints.

Examples

1. Create a table named NATION. Set four columns in the table and specify N_NATIONKEY as the primary key that enforces constraints to the table.

```
CREATE TABLE NATION (
    N_NATIONKEY  INTEGER,
    N_NAME       CHAR(25),
    N_REGIONKEY  INTEGER,
    N_COMMENT    VARCHAR(152),
    PRIMARY KEY (N_NATIONKEY)
);
```

2. Execute the following COPY statement to import data:

```
COPY NATION FROM stdin;
```

After the >> flag appears, enter the following rows of data in sequence:

```
0 'ALGERIA' 0 'haggle. carefully final deposits detect slyly agai'
1 'ARGENTINA' 1 'al foxes promise slyly according to the regular accounts. bold request
s alon'
2 'BRAZIL' 1 'y alongside of the pending deposits. carefully special packages are about
the ironic forges. slyly speci'
3 'CANADA' 1 'eas hang ironic, silent packages. slyly regular packages are furiously ov
er the tithes. fluffily bold'
\.
```

Note

When you copy the preceding data, replace the space between the two column values with Tab.

3. Execute the following statement to query the NATION table. The preceding data is imported.

```
SELECT * from NATION;
```

The following result is returned:

```
 n_nationkey |          n_name           | n_regionkey |
n_comment

-------------+---------------------------+-------------+------------------------------
----------------------------------------------------------------------------
           2 | 'BRAZIL'                  |           1 | 'y alongside of the pending de
posits. carefully special packages are about the ironic forges. slyly speci'
           3 | 'CANADA'                  |           1 | 'eas hang ironic, silent packa
ges. slyly regular packages are furiously over the tithes. fluffily bold'
           0 | 'ALGERIA'                 |           0 | ' haggle. carefully final depo
sits detect slyly agai'
           1 | 'ARGENTINA'               |           1 | 'al foxes promise slyly accord
ing to the regular accounts. bold requests alon'
(4 rows)
```

4. Execute the following COPY statement to insert a row of data that conflicts with the primary key:

```
COPY NATION FROM stdin;
```

After the >> flag appears, enter the following rows of data in sequence:

```
0 'GERMANY' 3 'l platelets. regular accounts x-ray: unusual, regular acco'
\.
```

Note

When you copy the preceding data, replace the space between the two column values with Tab.

A similar error message is returned:

```
ERROR:  duplicate key value violates unique constraint "nation_pkey"
DETAIL:  Key (n_nationkey)=(0) already exists.
CONTEXT:  COPY nation, line 1
```

5. Execute the following COPY ON CONFLICT statement to update the conflicting data:

```
COPY NATION FROM stdin DO  ON CONFLICT DO UPDATE;
```

After the >> flag appears, enter the following rows of data in sequence:

```
0 'GERMANY' 3 'l platelets. regular accounts x-ray: unusual, regular acco'
\.
```

Note

When you copy the preceding data, replace the space between the two column values with Tab.

No error message is returned. Execute the following statement to query the NATION table. A row of data with the primary key value of 0 is updated in the table.

```
SELECT * FROM NATION;
```

The following result is returned:

```
 n_nationkey |          n_name           | n_regionkey |
n_comment

-------------+---------------------------+-------------+------------------------------
---------------------------------------------------------------------------
           2 | 'BRAZIL'                  |           1 | 'y alongside of the pending de
posits. carefully special packages are about the ironic forges. slyly speci'
           3 | 'CANADA'                  |           1 | 'eas hang ironic, silent packa
ges. slyly regular packages are furiously over the tithes. fluffily bold'
           1 | 'ARGENTINA'               |           1 | 'al foxes promise slyly accord
ing to the regular accounts. bold requests alon'
           0 | 'GERMANY'                 |           3 | 'l platelets. regular accounts
x-ray: unusual, regular acco'
(4 rows)
```

6. Execute the following COPY ON CONFLICT statement to ignore insertion of the conflicting data:

```
COPY NATION FROM stdin DO ON CONFLICT DO NOTHING;
```

After the >> flag appears, enter the following rows of data in sequence:

```
1 'GERMANY' 3 'l platelets. regular accounts x-ray: unusual, regular acco'
\.
```

Note

When you copy the preceding data, replace the space between the two column values with Tab.

No error message is returned. Execute the following statement to query the NATION table. The row of data with the primary key value of 1 is not updated in the table.

```
SELECT * FROM NATION;
```

The following result is returned:

```
 n_nationkey |          n_name           | n_regionkey |
n_comment

-------------+---------------------------+-------------+------------------------------
---------------------------------------------------------------------------
           2 | 'BRAZIL'                  |           1 | 'y alongside of the pending de
posits. carefully special packages are about the ironic forges. slyly speci'
           3 | 'CANADA'                  |           1 | 'eas hang ironic, silent packa
ges. slyly regular packages are furiously over the tithes. fluffily bold'
           1 | 'ARGENTINA'               |           1 | 'al foxes promise slyly accord
ing to the regular accounts. bold requests alon'
           0 | 'GERMANY'                 |           3 | 'l platelets. regular accounts
x-ray: unusual, regular acco'
(4 rows)
```

7.Use MaxCompute foreign tables to access MaxCompute data

MaxCompute foreign-data wrapper (FDW) of AnalyticDB for PostgreSQL is designed based on the PostgreSQL FDW framework to access data that is stored in MaxCompute.

MaxCompute FDW allows you to synchronize data from MaxCompute to AnalyticDB for PostgreSQL. You can create the following three types of MaxCompute foreign tables by using MaxCompute FDW:

- Non-partitioned foreign tables, which are mapped to non-partitioned MaxCompute tables.
- Last-level partition foreign tables, which are mapped to the last-level partitions of partitioned MaxCompute tables.
- Partitioned foreign tables, which are mapped to partitioned MaxCompute tables.

Use MaxCompute FDW

1. Create the MaxCompute FDW extension in an AnalyticDB for PostgreSQL database.

```
CREATE EXTENSION odps_fdw ;
```

2. Grant the permission of using MaxCompute FDW to all database accounts.

```
GRANT USAGE ON FOREIGN DATA WRAPPER odps_fdw TO PUBLIC;
```

1. For a new AnalyticDB for PostgreSQL instance, the MaxCompute FDW extension is automatically created. In this case, you **can skip the preceding steps**.
2. For an existing AnalyticDB for PostgreSQL instance, you can use the **privileged database account** to connect to a specific database, and run the preceding commands to create the MaxCompute FDW extension and grant all database accounts the permission to use the extension.

Use a MaxCompute foreign table

Before you use a MaxCompute foreign table, you must perform the following operations:

- Create a MaxCompute server to specify the endpoint for accessing MaxCompute.
- Create a MaxCompute user mapping to specify the account that can access the created MaxCompute server.
- Create a MaxCompute foreign table to specify the MaxCompute table that you want to access.

1. 1. Create a MaxCompute server

1.1 Sample command

```
CREATE SERVER odps_serv                         -- The name of the MaxCompute server
  FOREIGN DATA WRAPPER odps_fdw
  OPTIONS (
    tunnel_endpoint '<odps tunnel endpoint>'    -- The endpoint of the MaxCompute Tunnel ser
vice
  );
```

1.2 Options

You need to specify only one of the tunnel_endpoint and odps_endpoint options when you create a MaxCompute server in AnalyticDB for PostgreSQL. The following table describes these options.

Option	Required	Description
tunnel_endpoint	No. We recommend that you specify this option.	The endpoint of the MaxCompute Tunnel service.
odps_endpoint	No.	The endpoint of the MaxCompute service.

Note

- When you create a MaxCompute server, you can specify one or both of these options. If both options are specified, the specified MaxCompute Tunnel endpoint takes precedence. If the MaxCompute Tunnel endpoint is not specified, the MaxCompute endpoint is used to route access requests to the corresponding MaxCompute Tunnel endpoint.
- We recommend that you use the MaxCompute Tunnel endpoint in the Alibaba Cloud classic network or a VPC.
- When you use the Tunnel endpoint to access data over the Internet, you are charged USD 0.1166 for each GB of data.

For more information about the MaxCompute endpoint, see Endpoints.

2. 2. Create a user mapping to a MaxCompute server

2.1 Sample command

```
CREATE USER MAPPING FOR { username | USER | CURRENT_USER | PUBLIC }
  SERVER odps_serv                                   -- The name of the MaxCompute server
  OPTIONS (
    id '<odps access id>',                                 -- The AccessKey ID of the accoun
t used to access the MaxCompute server
    key '<odps access key>'                                -- The AccessKey secret of the ac
count used to access the MaxCompute server
  );
```

Note

- **username** specifies the name of an existing account to map to the MaxCompute server.
- **CURRENT_USER** or **USER** specifies the name of the current user.
- If **PUBLIC** is specified, a public mapping is created. When no user-specific mapping is applicable, the public mapping is used.

2.2 Options

You must specify the type, the AccessKey ID, and the AccessKey secret of an AnalyticDB for PostgreSQL account that is used to access the MaxCompute server.

Option	Required	Description
`id`	Yes	The AccessKey ID of the account.
`key`	Yes	The AccessKey secret of the account.

3. 3. Create a MaxCompute foreign table

3.1 Sample command

```
CREATE FOREIGN TABLE IF NOT EXISTS table_name ( -- The name of the MaxCompute foreign table
    column_name data_type [, ... ]
)
  SERVER odps_serv                               -- The name of the MaxCompute server
  OPTIONS (
    project '<odps project>',                    -- The name of the MaxCompute project
    table '<odps table>'                         -- The name of the MaxCompute table
);
```

3.2 Options

You can create a MaxCompute foreign table after you create a MaxCompute server and a user mapping to the MaxCompute server. The following table describes the options.

Option	Required	Description
`project`	Yes	The MaxCompute project. A project is a basic unit of MaxCompute. Similar to a database or schema in a traditional database system, a project is used to isolate users and control access requests. For more information, see Project.

Option	Required	Description
`table`	Yes	The MaxCompute table. MaxCompute stores data in tables. For more information, see Table.
`partition`	No	The last-level partition in the partitioned MaxCompute table. Partitioning refers to dividing data in a table into independent parts based on partition keys. A partition key can be a single column or a combination of multiple columns. If a table is not partitioned, data is stored in the directory that stores the table. If a table is partitioned, each partition corresponds to a subdirectory in the directory that stores the table. In this case, data is stored in separate subdirectories. For more information about partitions, see Partition.

3.3 Types of foreign tables

MaxCompute FDW allows you to create the following three types of MaxCompute foreign tables based on the types of MaxCompute tables.

- Non-partitioned foreign tables

  A non-partitioned foreign table is mapped to a non-partitioned MaxCompute table. When you create a non-partitioned foreign table, you need only to specify valid values for the **project** and **table** options. You do not need to specify the **partition** option, or you can leave the **partition** option empty. Example:

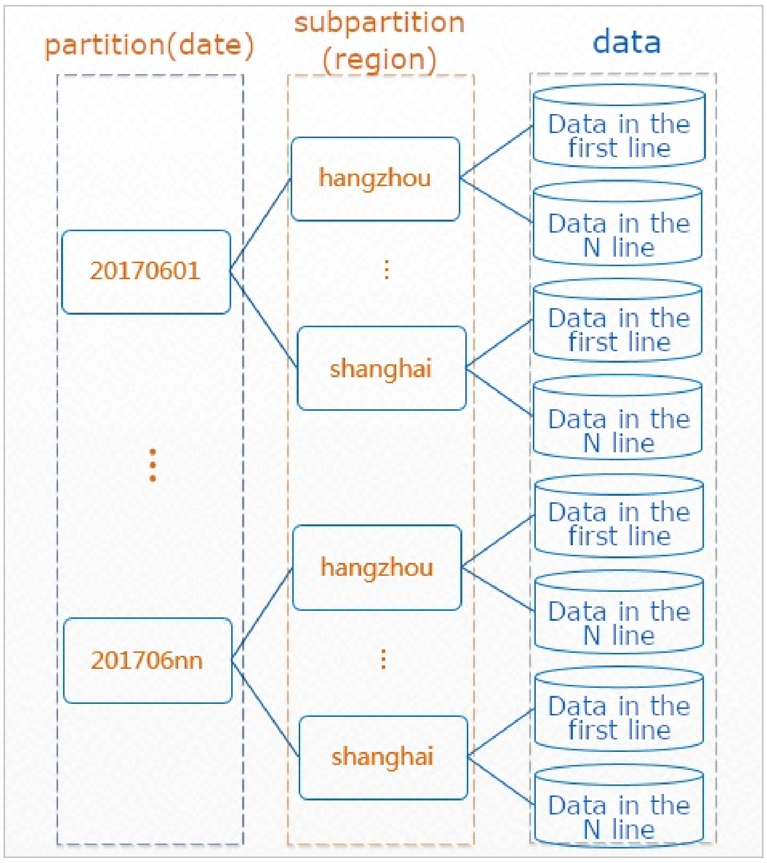
```
CREATE FOREIGN TABLE odps_lineitem (                -- The name of the MaxCompute foreign t
able
    l_orderkey      bigint,
    l_partkey       bigint,
    l_suppkey       bigint,
    l_linenumber    bigint,
    l_quantity      double precision,
    l_extendedprice double precision,
    l_discount      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 odps_serv                                -- The name of the MaxCompute server
OPTIONS (
  project 'odps_fdw',                             -- The name of the MaxCompute project
  table 'lineitem_big'                            -- The name of the MaxCompute table
);
```

- Last-level partition foreign tables

  A last-level partition foreign table is mapped to a last-level partition of a MaxCompute table. When you create a last-level partition foreign table, you must specify valid values for the **partition** option. If a MaxCompute table is partitioned in multiple levels, you can map the last-level partition foreign table only to a last-level partition of the MaxCompute table. You must set the **partition** option to the full path of the last-level partition.

  In the following example, a partitioned table that contains two levels of partitions in MaxCompute is created:

```
-- Create a partitioned table that contains two levels of partitions. The partition is ba
sed on the date and the subpartition is based on the region.
CREATE TABLE src (key string, value bigint) PARTITIONED BY (pt string,region string);
```



For example, assume that a MaxCompute table is partitioned in two levels and contains the 20170601 partition and the Hangzhou subpartition. To create a last-level partition foreign table that is mapped to the Hangzhou subpartition in an AnalyticDB for PostgreSQL database, the partition option must be set to pt=20170601,region=hangzhou.

```
CREATE FOREIGN TABLE odps_src_20170601_hangzhou (   -- The name of the MaxCompute foreign 
table
  key string,
  value bigint
) SERVER odps_serv                                   -- The name of the MaxCompute server
OPTIONS (
  project 'odps_fdw',                                -- The name of the MaxCompute project

  table 'src',                                       -- The name of the MaxCompute table
  partition 'pt=20170601,region=hangzhou'            -- The full path of the last-level pa
rtition
);
```

Note

i. Set the partition option in the key=value format. To specify a last-level partition in a multi-level partitioned table, separate the key-value pairs with commas (,). Do not include spaces in the partition option.

ii. You cannot map a last-level partition foreign table to a non-last-level partition of a MaxCompute table. In this example, you cannot set the partition option to `pt=20170601`, which indicates the path of the partition.

iii. Make sure that you set the partition option to the full path of a last-level partition. In this example, you cannot set the `partition` option to region=Hangzhou, which indicates the relative path of the last-level partition.

- Partitioned foreign tables

  A partitioned foreign table is mapped to a partitioned MaxCompute table. The following code shows how to use the preceding MaxCompute table src that contains two levels of partitions to create a partitioned foreign table. For more information, see Define table partitioning.

```
CREATE FOREIGN TABLE odps_src(                  -- The name of the MaxCompute foreign t
able
  key text,
  value bigint,
  pt text,                                      -- The partition key of the MaxCompute
foreign table
  region text                                   -- The subpartition key of the MaxCompu
te foreign table
) SERVER odps_serv
OPTIONS (
  project 'odps_fdw',                           -- The name of the MaxCompute project
  table 'src'                                   -- The name of the partitioned MaxCompu
te table
)
PARTITION BY LIST (pt)                          -- Use the pt column as the partition k
ey
SUBPARTITION BY LIST (region)                   -- Use the region column as the subpart
ition key
    SUBPARTITION TEMPLATE (                     -- The template of the subpartition
       SUBPARTITION hangzhou VALUES ('hangzhou'),
       SUBPARTITION shanghai VALUES ('shanghai')
    )
( PARTITION "20170601" VALUES ('20170601'),
  PARTITION "20170602" VALUES ('20170602'));
```

Note

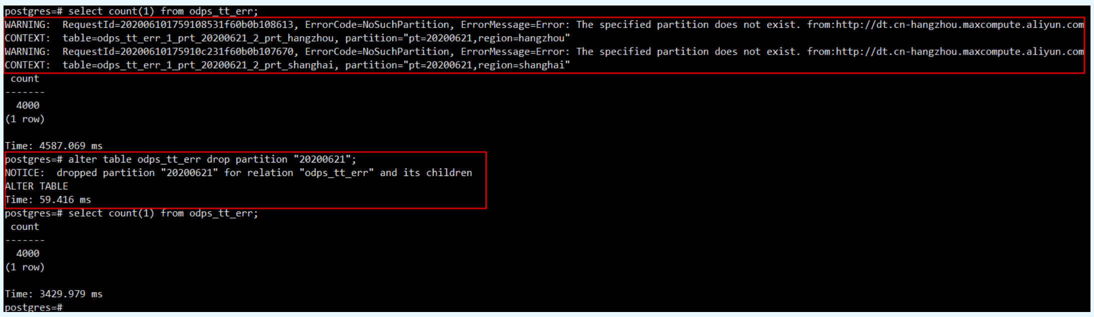
When you create a partitioned foreign table in an AnalyticDB for PostgreSQL database, make sure that the following requirements are met. These requirements are different from those on the creation of partitioned tables in MaxCompute.

i. Append the partition keys as fields to the end of other fields. If you create a multi-level partitioned foreign table, the partition key order, the level of partition keys, and the partition levels of the MaxCompute table must match one another.

ii. When you create a partitioned table, you must specify the partition key values. Use the **LIST** method to partition a table.

iii. When you create a partitioned foreign table, you do not need to specify the **partition** option. If the **partition** option is specified, the foreign table is created based on the specified last-level partition of the specified MaxCompute table instead of the entire MaxCompute table.

iv. If a partitioned foreign table contains partitions or subpartitions that have no matches in the specified MaxCompute table, an alert is triggered when you query the foreign table. In this case, you can delete the corresponding partition or subpartition from the foreign table. For more information, see the "**Delete partitions or subpartitions from a partitioned foreign table**" section in this topic.



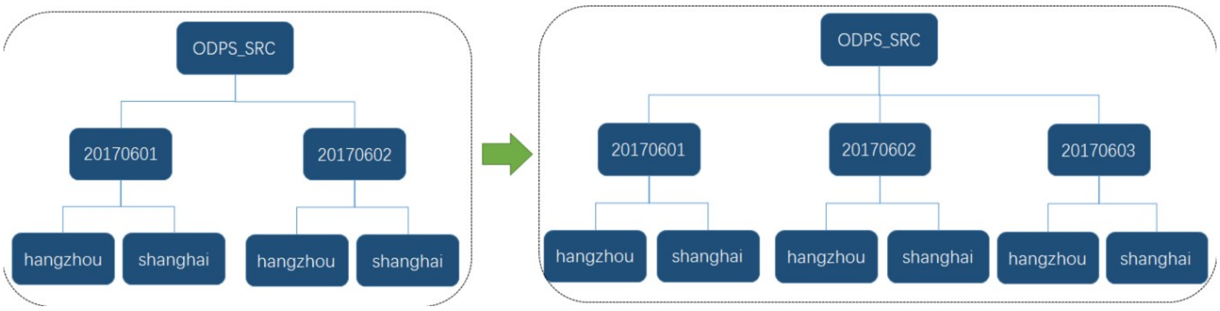
```
postgres=# select count(1) from odps_tt_err;
WARNING:  RequestId=202006101759108531f60b0b108613, ErrorCode=NoSuchPartition, ErrorMessage=Error: The specified partition does not exist. from:http://dt.cn-hangzhou.maxcompute.aliyun.com
CONTEXT:  table=odps_tt_err_1_prt_20200621_2_prt_hangzhou, partition="pt=20200621,region=hangzhou"
WARNING:  RequestId=20200610175910c231f60b0b107670, ErrorCode=NoSuchPartition, ErrorMessage=Error: The specified partition does not exist. from:http://dt.cn-hangzhou.maxcompute.aliyun.com
CONTEXT:  table=odps_tt_err_1_prt_20200621_2_prt_shanghai, partition="pt=20200621,region=shanghai"
 count
-------
  4000
(1 row)

Time: 4587.069 ms
postgres=# alter table odps_tt_err drop partition "20200621";
NOTICE:  dropped partition "20200621" for relation "odps_tt_err" and its children
ALTER TABLE
Time: 59.416 ms
postgres=# select count(1) from odps_tt_err;
 count
-------
  4000
(1 row)

Time: 3429.979 ms
postgres=#
```

3.4 Add partitions or subpartitions to a partitioned foreign table

In this example, the preceding partitioned foreign table odps_src is used.

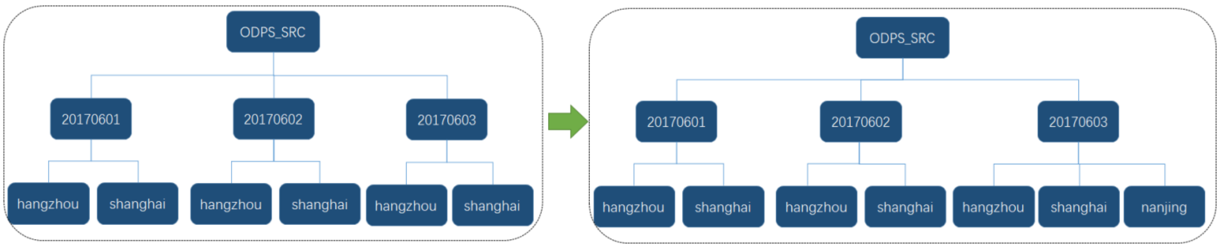
- Add a partition, as shown in the following figure.

```
-- Add a partition. The subpartitions are automatically created.
alter table odps_src add partition "20170603" values(20170603);
```



- Add a subpartition, as shown in the following figure.

```
-- Add a subpartition.
alter table odps_src alter partition "20170603" add partition "nanjing" values('nanjing');
```

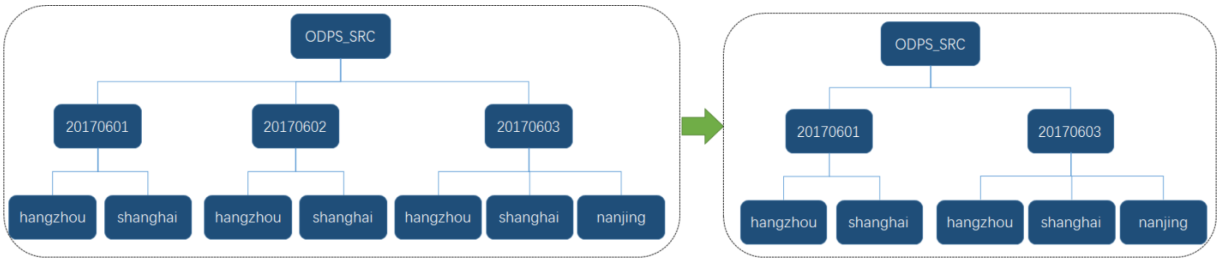


3.5 Delete partitions or subpartitions from a partitioned foreign table

In this example, the preceding partitioned foreign table odps_src is used.

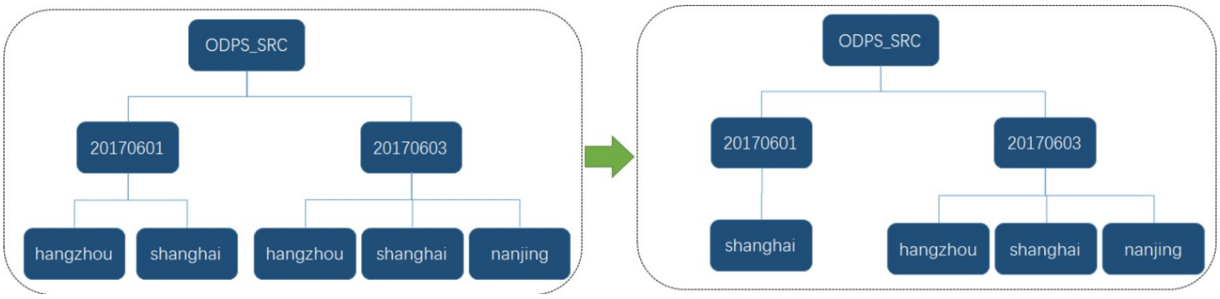
- Delete a partition, as shown in the following figure.

```
-- Delete a partition. The cascaded subpartitions are also deleted.
alter table odps_src drop partition "20170602";
```



- Delete a subpartition, as shown in the following figure.

```
-- Delete a subpartition.
alter table odps_src alter partition "20170601" drop partition "hangzhou";
```



Data types supported by MaxCompute foreign tables

The following table lists the data type mappings between MaxCompute and AnalyticDB for PostgreSQL. We recommend that you specify the data types of columns in a foreign table in an AnalyticDB for PostgreSQL database based on this table.

Note

AnalyticDB for PostgreSQL does not support data types that correspond to the STRUCT, MAP, and ARRAY data types that are supported by MaxCompute.

MaxCompute data type	AnalyticDB for PostgreSQL data type
BOOLEAN	bool
TINYINT	int2
SMALLINT	int2
INTEGER	int4
BIGINT	int8
FLOAT	float4
DOUBLE	float8
DECIMAL	numeric
BINARY	bytea
VARCHAR(n)	varchar(n)
CHAR(n)	char(n)
STRING	text
DATE	date
DATETIME	timestamp
TIMESTAMP	timestamp

Scenarios

You can execute foreign scans on AnalyticDB for PostgreSQL databases to scan MaxCompute foreign tables. Therefore, you can use the same query statements to query data in foreign tables and data in other tables in AnalyticDB for PostgreSQL. In this example, TPC Benchmark™H (TPC-H) queries are used to illustrate common scenarios where MaxCompute foreign tables are used.

Query a MaxCompute foreign table

TPC-H Q1 queries are used to aggregate and filter data in a single table. In this example, a Q1 query is performed on the MaxCompute foreign table odps_lineitem.

```
-- Create the MaxCompute foreign table odps_lineitem.
CREATE FOREIGN TABLE odps_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)
) server odps_serv
    options (
        project 'odps_fdw', table 'lineitem'
    );

-- TPC-H Q1
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    odps_lineitem
where
    l_shipdate <= date '1998-12-01' - interval '88' day --(3)
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;
```

Import MaxCompute data to an on-premises table

Perform the following steps to import data:

1. Create a MaxCompute foreign table in an AnalyticDB for PostgreSQL database.
2. Execute one of the following statements to import data to the new table:

```
-- INSERT statement
INSERT INTO <Destination on-premises table> SELECT * FROM <MaxCompute foreign table>;

-- CREATE TABLE AS statement
CREATE TABLE <Destination on-premises table> AS SELECT * FROM <MaxCompute foreign table>;
```

- **Example 1**: Execute the INSERT statement to import the data in odps_lineitem to an on-premises append-optimized column-oriented storage (AOCS) table.

```
-- Create an on-premises 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)
DISTRIBUTED BY (l_orderkey);

-- Import the data in odps_lineitem_orc to the on-premises AOCS table.
INSERT INTO aocs_lineitem SELECT * FROM odps_lineitem;
```

- **Example 2**: Execute the CREATE TABLE AS statement to import the data in odps_lineitem to an on-premises heap table.

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

Associate a MaxCompute foreign table with an on-premises table

In this example, a TPC-H Q19 query is performed on the MaxCompute foreign table odps_part that is associated with the on-premises AOCS table aocs_lineitem.

```
-- TPC-H Q19
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    aocs_lineitem,          -- The name of the on-premises AOCS table
    odps_part               -- The name of the MaxCompute foreign table
where
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#32'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 8 and l_quantity <= 8 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#41'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 15 and l_quantity <= 15 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#44'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 22 and l_quantity <= 22 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    );
```

Common errors in using MaxCompute foreign tables

Tunnel Common Errors

Notes

You can synchronize data from MaxCompute to MaxCompute foreign tables by using MaxCompute Tunnel. The synchronization performance is subject to the server resources and the outbound network bandwidth of MaxCompute Tunnel. Therefore, we recommend that you take note of the following items:

1. If you use only foreign tables, an AnalyticDB for PostgreSQL database can contain a maximum of five foreign tables.
2. If you associate multiple MaxCompute foreign tables, import the data in large MaxCompute foreign tables to on-premises tables and then associate the on-premises tables with small foreign tables to improve the performance.

8.Use OSS foreign tables to access OSS data

This topic describes how to use Object Storage Service (OSS) foreign tables to access OSS data. OSS foreign tables are developed based on the PostgreSQL Foreign Data Wrapper (FDW) framework to access OSS data for data analysis.

Overview

You can use OSS foreign tables to perform the following operations:

- Import OSS data to the internal row-oriented tables or column-oriented tables of the AnalyticDB for PostgreSQL instance for accelerated data analysis.
- Query and analyze large amounts of OSS data.
- Join OSS foreign tables to internal tables for data analysis.

OSS foreign tables allow you to access data objects in the ORC, Parquet, JSON, JSON Lines, and CSV formats. Access to GZIP- or standard Snappy-compressed CSV objects is also supported. You can partition an OSS foreign table based on one or more columns to filter out undesired partitions when you query a specific partition.

OSS data sources include business applications, log archives of Alibaba Cloud Log Service, and extract, transform, load (ETL) operations of Data Lake Analytics (DLA).

Differences between OSS foreign tables and OSS external tables

- AnalyticDB for PostgreSQL allows you to use OSS external tables to import and export data. However, OSS external tables do not meet the analysis requirements of large amounts of OSS data.
- OSS foreign tables are developed based on the PostgreSQL FDW framework and support ORC and CSV objects. Access to GZIP-compressed CSV objects is also supported. OSS foreign tables can be partitioned based on one or more fields. You can collect the statistics for OSS foreign tables so that the optimizer can generate an optimal execution plan.
- Foreign tables are superior to external tables in terms of performance, features, and stability. Therefore, the Greenplum community plans to replace external tables with foreign tables.

Use an OSS foreign table

To use an OSS foreign table, you must define **OSS FOREIGN TABLE** on **OSS FOREIGN SERVER** by using the **USER MAPPING** user. You can use CREATE USER MAPPING to specify the user, CREATE SERVER to specify the OSS server, and CREATE FOREIGN TABLE to specify the OSS foreign table.

For example, you can use the ossutil command line tool to view the following information about a TPC-H lineitem table in an OSS bucket. The following command provides an example on how to view the information by using ossutil:

```
ossutil ls oss://adbpg-tpch/data/tpch_data_10x/lineitem.tbl
```

In the command:

- **adbpg-tpch**: the name of the OSS bucket.
- **data/tpch_data_10x/...**: the path of the object in the bucket.

A similar list is returned:

```
LastModifiedTime                   Size(B)  StorageClass   ETAG
ObjectName
2020-03-12 09:29:48 +0800 CST    144144997      Standard   1F426F2FFC70A0262D2D69183BC3A0BD
-57   oss://adbpg-tpch/data/tpch_data_10x/lineitem.tbl.1
2020-03-12 09:29:58 +0800 CST    145177420      Standard   CFE2CFF1C8059547DC9F1711E77F74DD
-57   oss://adbpg-tpch/data/tpch_data_10x/lineitem.tbl.10
2020-03-10 21:23:24 +0800 CST    145355168      Standard   35C6227D1C29F1236A92A4D5D7922625
-57   oss://adbpg-tpch/data/tpch_data_10x/lineitem.tbl.11
... ...
```

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

Create an OSS server

To create an OSS server, define an OSS server you want to access. You must specify the `endpoint` parameter.

Syntax:

```
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 that contains the data object.
  );
```

The following table describes the values of OPTIONS. For more information, see CREATE SERVER.

Option	Type	Required	Description

Option	Type	Required	Description
oss endpoint	String	Yes	The endpoint of the OSS server. **Note** If you want to use an Alibaba Cloud server to access your AnalyticDB for PostgreSQL instance, use an internal endpoint to prevent incurring Internet traffic. An internal endpoint contains the keyword `internal`.
oss bucket	String	Optional	The bucket that contains the data object. You must create a bucket in OSS in advance. **Note** You must specify a bucket for either an OSS server or an OSS table. If you specify a bucket for both, the bucket value for the OSS table overwrites that for the OSS server.
Note • The following fault tolerance parameters can be specified when you access OSS. You can retain the default values. • If the following parameters use default values, a timeout is triggered if the transmission rate is less than 1 KB/s for consecutive 1,500 seconds. For more information, see Error handling.			
speed_limit	Numeric value	Optional	Specifies the minimum transmission rate.
speed_time	Numeric value	Optional	Specifies the maximum duration for maintaining the minimum transmission rate.

Option	Type	Required	Description
connect_timeout	Numeric value	Optional	Specifies the connection timeout period.
dns_cache_timeout	Numeric value	Optional	Specifies the timeout period for DNS resolution.

Create a user mapping to the OSS server

After you create an OSS server, you must create a user that accesses the OSS server. To define the mapping from an AnalyticDB for PostgreSQL user to the OSS server user, you can create an OSS user mapping.

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
```

Parameter options

The following table describes the parameter options. For more information, see CREATE USER MAPPING.

Option	Required	Description
`id`	Yes	Specifies the account ID used to access OSS.
`key`	Yes	Specifies the AccessKey secret of the account used to access OSS.
`username`	Optional	Specifies the name of an existing user that is mapped to the foreign server.
`CURRENT_USER` or `USER`	Optional	Specifies that the mapping is created or deleted for the current user.

Option	Required	Description
`PUBLIC`	Optional	Specifies all roles, including roles to create.

Examples

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

• Create an OSS foreign table

After you create an OSS server and a user used to access the OSS server, you can define an OSS foreign table. OSS foreign tables allow you to access data objects in a variety of formats to meet the requirements of different business scenarios.

Objects in the following formats are supported:

- Uncompressed CSV, TEXT, JSON, and JSON Lines objects.
- GZIP- and standard Snappy-compressed CSV and TEXT objects. GZIP-compressed JSON and JSON Lines objects.
- ORC binary objects. For more information about the data type mapping from ORC objects to AnalyticDB for PostgreSQL files, see

  Data type mappings between ORC objects and AnalyticDB for PostgreSQL files.
- Parquet binary objects. For more information about the data type mapping from Parquet objects to AnalyticDB for PostgreSQL files, see

  Data type mappings between Parquet objects and AnalyticDB for PostgreSQL files.

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 ]
```

Parameter options

The following two tables describe the options. For more information, see CREATE FOREIGN TABLE.

Common options

Option	Type	Required	Remarks
filepath	String	Yes. Select one of the three parameters. Note: The three parameters specify the path of the object in the OSS bucket.	This path directs to a single object.
prefix	String		None
dir	String		None
bucket	String	Optional	You must specify a bucket for either an OSS server or an OSS table. If you specify a bucket for both, the bucket value for the OSS table overwrites that for the OSS server.
format	String	Yes	The format of the object. Valid values: • CSV. • TEXT. • ORC. • PARQUET. • JSON. For more information about JSON standards, see Introducing JSON. • JSONLINE. You can view JSON Lines as line feed-delimited JSON. All data that can be read by using JSON Lines can be read by using JSON, but not vice versa. We recommend that you use JSON Lines if possible. For more information about JSON Lines standards, see JSONLINE.

Options for CSV and TEXT

Note

Unless otherwise specified, the options described in the following table apply only to the CSV and TEXT formats. The parameter configurations are invalid for formats such as ORC and Parquet.

Option	Type	Required	Default value	Remarks
filetype	String	Optional	plain	Valid values: • plain: The system reads only the raw binary data. • gzip: The system reads the raw binary data and decompresses the package by using GZIP. • snappy: The system reads raw binary data and decompresses the package by using standard Snappy. You can read only standard Snappy-compressed objects. You cannot read objects compressed by using Hadoop-Snappy. For more information, see snappy/format_description.txt. You can also use this parameter to specify the compression type of the input object in the JSON or JSON Lines format. Valid values: plain and gzip.
log_errors	Boolean	Optional	false	Specifies whether to record errors in log files. For more information, see Error tolerance mechanism.
segment_reject_limit	Numeric value	Optional	None	Specifies the maximum 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. Examples: • segment_reject_limit = '10' indicates that execution is aborted when the number of error lines in a segment exceeds 10. • segment_reject_limit = '10' indicates that execution is aborted when the number of error lines in a segment is more than 10% of the processed lines.
The following table describes the options for formatting. For more information, see COPY.				
header	Boolean	Optional	false	Specifies whether the source file contains the header row. This option applies only to the CSV format.

Option	Type	Required	Default value	Remarks
delimiter	String	Optional	• Default value for TEXT objects: the Tab key. • Default value for CSV objects: a comma (,).	The field delimiter. Only single-byte characters are allowed.
quote	String	Optional	Double quotation marks (").	The column quotation marks. • This parameter applies only to the CSV format. • Only single-byte characters are allowed.
escape	String	Optional	By default, the value is the same as the value of the quote parameter.	Specifies the character that appears before a character that is the same as the value of the quote parameter. • Only single-byte characters are allowed. • This parameter applies only to the CSV format.
null	String		• Default value for TEXT objects: \N • Default value for CSV objects: spaces that are not enclosed in double quotation marks(").	Specifies the empty string for a file.
encoding	String	Optional	By default, the encoding format on the client is used.	Specifies the encoding format of the data object.
force_not_null	Boolean	Optional	false	If the parameter is set to true, the values of specified columns are not matched against the empty string.

Option	Type	Required	Default value	Remarks
force_null	Boolean	Optional	false	• If this parameter is set to true, the column values that match the empty string are returned as NULL even if the values are quoted. • If this parameter is not specified, only unquoted column values that match the empty string are returned as NULL.

Examples

```
CREATE FOREIGN TABLE x(i int, j int)
SERVER oss_serv OPTIONS (format 'jsonline')
PARTITION BY LIST (j) (
    VALUES('20181218'),
    VALUES('20190101')
);
```

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>');`

Fault tolerance mechanism

When you create an OSS foreign table, you can set the `log_errors` and `segment_reject_limit` parameters to prevent unexpected exits due to error lines in the raw data during a scan of the OSS foreign table. The following list describes these parameters:

- `log_errors` : specifies whether to record the information of error lines.
- `segment_reject_limit` : specifies the maximum allowable percentage of all parsed lines that can be error lines.

  Note

  Only OSS foreign tables in the CSV and TEXT formats support the fault tolerance mechanism.

To implement the error tolerance mechanism, perform the following steps:

1. Create an OSS foreign table based on the FDW feature for 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 error lines.
             segment_reject_limit '10', -- The number of error lines cannot exceed 10.
Otherwise, the system returns an error and exits.
             dir 'error_sales/',         -- Specify the OSS object directory that the fo
reign table matches.
             format 'csv',               -- Parse objects in the CSV format.
             encoding 'utf8');           -- Specify the encoding format.
```

2. Scan the created foreign table.

   Note

   To show the fault tolerance effect, three error lines are added to the OSS objects.

   The following statement provides an example on how to execute the query:

```
SELECT * FROM oss_error_sales ;
```

   A similar output is returned:

```
 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.

   The following statement provides an example on how to execute the query:

```
SELECT * FROM gp_read_error_log('oss_error_sales');
```

   A similar output is returned:

```
           cmdtime             |     relname     |        filename         | linenum | b
ytenum |                          errmsg                                | rawdata | rawbytes
------------------------------+-----------------+-------------------------+---------+--
-------+-----------------------------------------------------------+---------+---------
-
 2020-04-22 19:37:35.21125+08 | oss_error_sales | error_sales/sales.2.csv |       2 |
| invalid byte sequence 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 sequence 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 sequence for encoding "UTF8": 0xed 0xab 0xad |         | \x
(3 rows)
```

4. Delete the log of error lines.

   The following statement provides an example on how to execute the query:

```
SELECT gp_truncate_error_log('oss_error_sales');
```

   A similar output is returned:

```
 gp_truncate_error_log
-----------------------
 t
(1 row)
```

Use an OSS foreign table

1. To import OSS data to an internal table, perform the following steps:

   i. Distribute data evenly to multiple objects in OSS.

   Note

   - All compute nodes of an AnalyticDB for PostgreSQL instance read data in parallel from the data objects stored in OSS based on the round-robin algorithm.
   - We recommend that you use the same data encoding format for data objects and databases to simplify the encoding process and improve efficiency. The default database encoding format is UTF-8.
   - Multiple CSV and TEXT objects can be read in parallel. By default, four objects can be read in parallel.
   - To maximize the read efficiency, we recommend that you set the number of objects that can be read in parallel to an integer multiple of the number of compute node cores. The number of compute node cores is the product of the number of compute nodes and the number of cores per compute node.
   - If the number of objects is small, we recommend that you split the source objects into multiple objects so that multiple nodes can scan the objects in parallel. For more information, see Split files.

   ii. Create an OSS foreign table for your AnalyticDB for PostgreSQL instance.

iii. Execute one of the following statements to import data:

- INSERT INTO statement

```
INSERT INTO <Destination on-premises table> SELECT * FROM <OSS foreign table>;
```

- CREATE TABLE AS statement

```
CREATE TABLE <Destination on-premises table> AS SELECT * FROM <OSS foreign table>
;
```

Examples:

- Example 1: Use the INSERT statement to import data from oss_lineitem_orc to an internal AOCS table

  Create an internal table named aocs_lineitem. Example:

```
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 data from oss_lineitem_orc to the aocs_lineitem internal table. Example:

```
INSERT INTO aocs_lineitem SELECT * FROM oss_lineitem_orc;
```

- Example 2: Use the CREATE TABLE AS statement to import data from oss_lineitem_orc to an internal heap table

```
CREATE TABLE heap_lineitem AS SELECT * FROM oss_lineitem_orc DISTRIBUTED BY (l_orderk
ey);
```

2. Query and analyze OSS data.

   You can query an OSS foreign table in the same way as you query an internal table. The following list describes the common query scenarios:

   - Data filtering based on key-value pairs

```
SELECT * FROM oss_lineitem_orc WHERE l_orderkey = 14062498;
```

   - Data aggregation

```
SELECT count(*) FROM oss_lineitem_orc WHERE l_orderkey = 14062498;
```

- A combination of the filter, group, and limit features

```
SELECT l_partkey, sum(l_suppkey)
  FROM oss_lineitem_orc
 GROUP BY l_partkey
 ORDER BY l_partkey
 limit 10;
```

3. Join OSS foreign tables to internal tables for data analysis.

   Join internal AOCS table aocs_lineitem to OSS foreign tables for a TPC-H Q3 query. Example:

```
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                                    -- Internal 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;
```

Use a partitioned OSS foreign table

In addition to OSS foreign tables, AnalyticDB for PostgreSQL supports partitioned OSS foreign tables. If you use partition key columns in a WHERE clause to query data, you can reduce the amount of data to be pulled from OSS.

To use the partitioning feature of FDW-based OSS foreign tables in AnalyticDB for PostgreSQL, make sure that the following requirements are met: The data of a partition in a partitioned OSS foreign table must be stored in the `oss://bucket/partcol1=partval1/partcol2=partval2/` directory of the OSS server. In this directory, `partcol1` and `partcol2` indicate partition key columns, and `partval1` and `partval2` indicate the partition key column values that define the partition.

Syntax

- Create a partitioned OSS foreign table

  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 syntax of partitioning standard partitioned tables, see Table partitioning.

Note

FDW-based OSS foreign tables in AnalyticDB for PostgreSQL support only list partitioning.

- Delete a partitioned OSS foreign table

  You can execute the DROP FOREIGN TABLE statement to delete a regular foreign table. You can also execute this statement to delete a partitioned OSS foreign table.

- Adjust the structure of a partitioned foreign table

  You can execute the ALTER TABLE statement to adjust the structure of a partitioned foreign table. You can add and delete partitions. For more information about the syntax, see ALTER TABLE. The following content provides examples on how to create and delete a partition. Execute the following statement to create the ossfdw_parttable table:

```
CREATE FOREIGN TABLE ossfdw_parttable(
  key text,
  value bigint,
  pt text,                                         -- The partition key of the foreign tab
le.
  region text                                      -- The subpartition key of the foreign
table.
)
SERVER oss_serv
OPTIONS (dir 'PartationDataDirInOss/', format 'jsonline')
PARTITION BY LIST (pt)                             -- Use the "pt" column as the partition
key
SUBPARTITION BY LIST (region)                      -- Use the "region" column as the subpa
rtition key
    SUBPARTITION TEMPLATE (                        -- The template of the subpartition
       SUBPARTITION hangzhou VALUES ('hangzhou'),
       SUBPARTITION shanghai VALUES ('shanghai')
    )
( PARTITION "20170601" VALUES ('20170601'),
  PARTITION "20170602" VALUES ('20170602'));
```

  Execute the ALTER TABLE statement to add a partition to the table. When you perform this operation, a subpartition is automatically created for the new partition based on the subpartition template that is defined in the preceding code block.

```
ALTER TABLE ossfdw_parttable ADD PARTITION VALUES ('20170603');
```

  You can also create a subpartition for an existing partition.

```
ALTER TABLE ossfdw_parttable ALTER PARTITION FOR ('20170603') ADD PARTITION VALUES('nanji
ng');
```

  Delete a partition.

```
ALTER TABLE ossfdw_parttable DROP PARTITION FOR ('20170601');
```

  Delete a subpartition.

```
ALTER TABLE ossfdw_parttable ALTER PARTITION FOR ('20170602') DROP PARTITION FOR ('hangzh
ou');
```

When you create or add a foreign table partition, you can customize the partition instead of using a template. The following code block provides an example:

```
CREATE FOREIGN TABLE ossfdw_parttable(
  key text,
  value bigint,
  pt text,                                    -- The partition key of the foreign tab
le.
  region text                                 -- The subpartition key of the foreign
table.
)
SERVER oss_serv
OPTIONS (dir 'PartationDataDirInOss/', format 'jsonline')
PARTITION BY LIST (pt)                        -- Use the "pt" column as the partition
key
SUBPARTITION BY LIST (region)                 -- Use the "region" column as the subpa
rtition key
(
    -- The following two partitions have different subpartitions:
    VALUES('20181218')
    (
        VALUES('hangzhou'),
        VALUES('shanghai')
    ),
    VALUES('20181219')
    (
        VALUES('nantong'),
        VALUES('anhui')
    )
);
```

Add a partition to the table. You must specify subpartitions for the new partition because no subpartition templates are defined. Example:

```
ALTER TABLE ossfdw_parttable ADD PARTITION VALUES ('20181220')
(
    VALUES('hefei'),
    VALUES('guangzhou')
);
```

Scenarios

Partitioned foreign tables are commonly used to access logs shipped by Log Service. For more information, see Use FDW-based OSS foreign tables to access logs shipped by Log Service. You can use a similar method to organize directories of the OSS server when you write data from business applications to OSS, and then create a partitioned OSS foreign table.

Examples

The following SQL statements are used to 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')
)
```

Use OSS foreign tables to export data

AnalyticDB for PostgreSQL allows you to export data to OSS by using foreign tables. OSS foreign tables support export of data objects in multiple formats for different business scenarios.

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

Note

AnalyticDB for PostgreSQL does not allow data export to partitioned foreign tables.

Naming conventions of exported objects

During export, multiple segments are used to export data to the same directory in parallel. Therefore, the name of the object exported to OSS uses the following rule:

```
{tablename | prefix } _{timestamp}_{random_key}_{seg}{segment_id}_{fileno}.{ext}[.gz]
```

- {tablename | prefix }: the name of the object after export. If `prefix` is used, the specified `prefix` is used as the prefix. If the `dir` option is used, the name of the OSS foreign table is used as the prefix.
- {timestamp}: the timestamp when the data is exported. Format: `YYYYMMDDHH24MISS`.

- {random_key}: the random key value.
- {seg}{segment_id}: {seg} specifies the segment. {segment_id} specifies the segment ID. For example, seg1 specifies that the object is exported from Segment 1.
- {fileno}: the SN of the object segment, which starts from 0.
- {ext}: the format of the object after export. For example, ".csv" corresponds to "csv" set for the `format` option, ".txt" to "text", and ".orc" to "orc".
- [.gz]: specifies that the object after export is a GZIP-compressed object.
    - Create a foreign table. Set the format to CSV. Use GZIP for compression, and use dir to specify the path.

      ```
      CREATE FOREIGN TABLE fdw_t_out_1(a int)
      SERVER oss_serv
      OPTIONS (format 'csv', filetype 'gzip', dir 'test/');
      ```

    - Specify the name of the OSS object after export. Example:

      ```
      fdw_t_out_1_20200805110207_1718599661_seg-1_0.csv.g
      ```

    - Create a foreign table, set the format to orc, and then use prefix to specify the prefix of the object after export.

      ```
      CREATE FOREIGN TABLE fdw_t_out_2(a int)
      SERVER oss_serv
      OPTIONS (format 'orc', prefix 'test/my_orc_test');
      ```

    - The name of the OSS object after export. Example:

      ```
      my_orc_test_20200924153043_1737154096_seg0_0.orc
      ```

Parameter options

For more information about the options of the AnalyticDB for PostgreSQL foreign table when you use the foreign table to export data, see Create an OSS foreign table.

In addition, when you use a foreign table to export data, you can use specific options.

Common options

Option	Type	Unit	Required	Default value	Remarks
fragment_size	Numeric value	MB	No	256	The size of the fragment when data is exported to OSS. If the size of the data written to the object exceeds the specified fragment_size value, data in excess of this value is distributed to a new segment.

Note

- Each fragment is an object fragment during export. Each fragment is complete and independent and therefore can be parsed by the foreign table of AnalyticDB for PostgreSQL. A row is not stored across fragments.
- The size of a fragment does not strictly follow the specified fragment_size value. In most cases, the actual size is slightly greater than the specified size.
- **If you do not have special requirements, you can retain the default value without changing the fragment_size value.**

An example of how to use fragment_size:

1. Create a foreign table whose format is CSV. The directory of the object after export is *test/lineitem/*. Set framgment_size to 512 MB.

```
CREATE FOREIGN TABLE oss_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)
) server oss_serv
    options (
        dir 'test/lineitem/',
        format 'csv',
        fragment_size '512' -- If the size of a fragment exceeds 512 MB, a new segment
is generated.
    );
```

2. Write data from the internal table to the foreign table.

```
INSERT INTO oss_lineitem SELECT * FROM lineitem;
```

3. After export, view objects in OSS. The sizes of most object fragments are slightly greater than 512 MB.

   Command:

```
ossutil -e endpoint -i id -k key ls oss://bucket/test/lineitem
```

   A similar output is returned:

```
LastModifiedTime                   Size(B)  StorageClass   ETAG
ObjectName
2020-09-24 14:12:01 +0800 CST    536875660      Standard   ED6C68093E738D09B1386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg15_
7.csv
2020-09-24 14:12:27 +0800 CST    536875604      Standard   FD25FA7C7109ABCDCB386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg15_
8.csv
2020-09-24 14:12:53 +0800 CST    536875486      Standard   7C3EDE6AFE354190E5386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg15_
9.csv
2020-09-24 14:09:07 +0800 CST    536875626      Standard   48B38E65A5BB8B5B03386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg1_0
.csv
2020-09-24 14:09:32 +0800 CST    536875858      Standard   AF5525D81166F02D1C386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg1_1
.csv
2020-09-24 14:13:08 +0800 CST    235457368      Standard   BF1FC0B81376AE14F4386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg1_1
0.csv
2020-09-24 14:09:56 +0800 CST    536875899      Standard   20C824EBCAE2C5DB34386C5F0000
0000      oss://adbpg-tpch/test/lineitem/oss_lineitem2_20200924140843_1702924182_seg1_2
.csv
...
```

Command:

```
ossutil -e endpoint -i id -k key du oss://adbpg-tpch/test/lineitem/
```

A similar output is returned:

```
storage class   object count            sum size(byte)
----------------------------------------------------------
Standard       176                  89686183118
----------------------------------------------------------
total object count: 176                  total object sum size: 89686183118
total part count:   0                           total part sum size:   0

total du size(byte):89686183118

0.051899(s) elapsed
```

CSV and TEXT options

Notice

Unless otherwise specified, the options described in the following table apply only to the CSV and TEXT formats. The option configurations are invalid for the ORC format.

Option	Type	Unit	Required	Default value	Remarks
gzip_level	Numeric value	None	No	1	The compression level of GZIP used when you export CSV or TEXT data to OSS. The highest compression level is 9.
force_quote_all	Boolean	None	No	false	Specifies whether to force reference all fields when you export CSV data. force_quote_all is valid only for the CSV format.

Note

- gzip_level takes effect only when filetype is set to `gzip`.
- gzip_level A higher GZIP compression level results in smaller sizes of generated data objects and longer export time.
- However, according to test results, a greater gzip_level value does not result in smaller file sizes, but does increase the export time. If you do not have special requirements, we recommend that you use the default value of gzip_level, which is 1.

The following procedure provides an example on how to use gzip_level:

1. Create a foreign table whose format is CSV. The directory of the object after export is *test/lineitem2/*. Set gzip_level to 9.

```
CREATE FOREIGN TABLE oss_lineitem3 (
    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
    options (
        dir 'test/lineitem2/',
        format 'csv', filetype 'gzip' ,gzip_level '9'
    );
```

2. Write data from the internal table to the foreign table.

   ```
   INSERT INTO oss_lineitem SELECT * FROM lineitem;
   ```

3. After export, you can view the objects in OSS. The total size of the objects is 23 GB (25,141,481,880 bytes), which is far less than the fragment_size value of 83 GB (89,686,183,118 bytes) specified in the example.

   Command:

   ```
   ossutil -e endpoint -i id -k key ls oss://bucket/test/lineitem2
   ```

   A similar output is returned:

```
2020-09-24 15:07:49 +0800 CST    270338060      Standard   9B1F7D7CB0748391C5456C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg8_
3.csv.gz
2020-09-24 15:10:08 +0800 CST    270467652      Standard   17DE92CAE57F64F550466C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg8_
4.csv.gz
2020-09-24 15:12:00 +0800 CST    219497966      Standard   FCFE4E457C0F6942C0466C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg8_
5.csv.gz
2020-09-24 15:01:10 +0800 CST    270617343      Standard   13AD24EF528ECDF836446C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg9_
0.csv.gz
2020-09-24 15:03:18 +0800 CST    270377032      Standard   759DBF9B999F8609B6446C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg9_
1.csv.gz
2020-09-24 15:05:28 +0800 CST    270284091      Standard   F9896A5CFF554F2838456C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg9_
2.csv.gz
2020-09-24 15:07:43 +0800 CST    270350284      Standard   C120BE98B47DAD5EBF456C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg9_
3.csv.gz
2020-09-24 15:10:04 +0800 CST    270477777      Standard   69B9B1E854B626364C466C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg9_
4.csv.gz
2020-09-24 15:12:00 +0800 CST    219358236      Standard   A4EB5DFFBD67AF6BC0466C5F0000
0000      oss://adbpg-tpch/test/lineitem2/oss_lineitem3_20200924145900_1220405754_seg9_
5.csv.gz
....

Object Number is: 96
```

Command:

```
ossutil -e endpoint -i id -k key du oss://adbpg-tpch/test/lineitem/
```

A similar output is returned:

```
storage class   object count            sum size(byte)
----------------------------------------------------------
Standard       96                   25141481880
----------------------------------------------------------
total object count: 96                   total object sum size: 25141481880
total part count:   0                           total part sum size:   0

total du size(byte):25141481880

0.037620(s) elapsed
```

The following procedure provides an example on how to use force_quote_all:

1. Create a foreign table whose format is CSV. The directory of the object after export is *test/lineitem/*. Set force_quote_all to true.

```
CREATE FOREIGN TABLE foreign_x (
  a int, b text
) server oss_serv
    options (
        dir 'test/x/',
        format 'csv', force_quote_all 'true'
    );
```

2. Write data from the internal table to the foreign table.

```
INSERT INTO foreign_x values(1, 'a'), (2, 'b');
```

3. After export, view the object content in OSS. The value of each column is referenced by double quotation marks (").

   Command:

```
cat foreign_x_20200923173618_447789894_seg4_0.csv
```

   A similar output is returned:

```
"1","a"
```

   Command:

```
cat foreign_x_20200923173618_447789894_seg5_0.csv
```

   A similar output is returned:

```
"2","b"
```

ORC option

Notice

Unless otherwise specified, the options described in the following table apply only to ORC. The option configurations are invalid for the CSV or TEXT format.

Option	Type	Unit	Required	Default value	Remarks
orc_stripe_size	Numeric value	MB	No	64	The size of each stripe in a single ORC object. You can specify the size based on which to export data to an ORC object.

Note

- A smaller orc_stripe_size value results in larger ORC objects and longer export time when the amount of data is the same.
- If you do not have special parameters, you can use the default value 64 MB without changing the orc_stripe_size value.

The following procedure provides an example on how to use orc_stripe_size:

1. Create a foreign table whose format is ORC. The directory of the object after export is test/lineitem/. Set orc_stripe_size to 128 MB.

   ```
   CREATE FOREIGN TABLE oss_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)
   ) server oss_serv
   options ( dir 'test/lineitem/', format 'orc', orc_stripe_size '128');
   ```

2. Write data from the internal table to the foreign table.

   ```
   INSERT INTO oss_lineitem SELECT * FROM lineitem;
   ```

3. After export, view the meta information of an exported ORC object. The size of the stripe in the ORC object is about 128 MB.

Examples

1. Create a foreign table

   Create a foreign table whose format is CSV. The directory of the object after export is tt_csv.

   ```
   CREATE FOREIGN TABLE foreign_x (i int, j int)
   SERVER oss_serv
   OPTIONS (format 'csv', dir 'tt_csv/');
   ```

2. Export data

   You can write data to a foreign table in the same manner as you write data to a standard table. You can use the INSERT INTO statement to write data.

   - Batch export data to the OSS foreign table. We recommend that you use this method.

```
INSERT INTO foreign_x SELECT * FROM local_x;
```

- Insert the list of values. We recommend that you do not use this method.

```
INSERT INTO foreign_x VALUES (1,1), (2,2), (3,3);x;
```

Split objects

FDW-based OSS foreign tables support multi-node parallel scans. If the number of objects is small, we recommend that you split the source objects into multiple objects so that multiple nodes can scan the objects in parallel.

For example, in Linux, you can run the **split** command to split objects in the `TEXT` or `CSV` format into multiple objects.

Note

A row in an object cannot be split into different objects. Therefore, objects must be split by rows.

- Obtain the number of rows in the current object.

```
wc -l csv_file
```

- Split the object into small objects based on the specified number of rows. N specifies the number of rows in each small object.

```
split -l N csv_file
```

Collect statistics for foreign tables

The data of OSS foreign tables is stored in OSS. By default, the statistics for foreign tables are not collected. If up-to-date statistics are unavailable, the query optimizer may generate inefficient query plans for complex queries such as queries on joined tables. You can use the ANALYZE statement to update statistics. You can use the following execution process:

- 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 for an OSS foreign table.

```
ANALYZE <Table name>;
```

- Use the EXPLAIN statement to view the execution plan after the ANALYZE statement is executed.

```
EXPLAIN <Table name>;
```

View execution plans

To view the execution plan of an OSS foreign table, execute the following statements:

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

Note

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

View the object information of a specified OSS foreign table

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

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

Data type mappings between ORC objects and AnalyticDB for PostgreSQL files

The following table describes the data type mappings between ORC objects 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

AnalyticDB for PostgreSQL data type	ORC data type
date	DATE
int2[]	LIST(SHORT)
int4[]	LIST(INT)
int8[]	LIST(LONG)
float4[]	LIST(FLOAT)
float8[]	LIST(DOUBLE)

Note

ORC data of the LIST type can be converted to only one-dimensional arrays in AnalyticDB for PostgreSQL.

Data type mappings between Parquet objects and AnalyticDB for PostgreSQL files

Objects in the Parquet format support the following data types: primitive types and logical types. We recommend that you use the following data types supported by AnalyticDB for PostgreSQL to map the data types supported by Parquet objects when you create a table. Otherwise, the data types are converted.

Note

Two nested data types in Parquet are not supported: ARRAY and MAP.

The following table describes the data type mappings that apply when Parquet objects do not contain data of logical types.

AnalyticDB for PostgreSQL data type	Parquet data type
bool	BOOLEAN
integer	INT32

AnalyticDB for PostgreSQL data type	Parquet data type
bigint	INT64
timestamp	INT96
float4	FLOAT
float8	DOUBLE
bytea	BYTE_ARRAY
bytea	FIXED_LEN_BYTE_ARRAY

The following table shows the data type mappings that apply when Parquet objects contain data of logical types.

AnalyticDB for PostgreSQL data type	Parquet data type
text	STRING
date	DATE
timestamp	TIMESTAMP
time	TIME
interval	INTERVAL
numeric	DECIMAL
smallint	INT(8)/INT(16)
integer	INT(32)
bigint	INT(64)

AnalyticDB for PostgreSQL data type	Parquet data type
bigint	UINT(8/16/32/64)
json	JSON
jsonb	BSON
uuid	UUID
text	ENUM

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

Partitioned FDW-based OSS foreign tables in AnalyticDB for PostgreSQL are widely used to access logs shipped by 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?

1. To create a partitioned OSS foreign table based on the data shipped from Log Service to OSS, set Shard Format in the OSS LogShipper dialog box. For more information about the semantic information of configuration items, see Ship log data from Log Service to OSS. Log objects generated within the same month are stored in the same OSS directory. The configuration generates the following directory structure in OSS:

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

2. Then, create the following partitioned OSS foreign table based on the columns contained in the files shipped 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')
)
```

3. Finally, implement the required analysis logic on the partitioned foreign table. For example, you can query the number of all user logons in February 2020.

   The following statement provides a sample request:

```
EXPLAIN SELECT uid, count(uid) FROM userlogin WHERE "date" = 202002 GROUP BY uid;
```

   A similar output is returned:

```
                                                                                    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 row
s=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
                     ->t;  Append  (cost=0.00..100.10 rows=333334 width=4)
                           ->  Foreign Scan on userlogin_1_prt_1  (cost=0.00..100.10 ro
ws=333334 width=4)
                                 Filter: (date = 202002)
                                 Oss Url: endpoint=oss-cn-hangzhou-zmf-internal.aliyunc
s.com bucket=adbpg-regress dir=adbpgossfdw/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 to reduce the amount of data to read and maximize query efficiency.

Common errors

When the system scans an OSS foreign table, a similar error is returned:

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

For more information about error types and how to handle errors, see Handle OSS errors.

In the error message:

- StatusCode: the HTTP status code.
- ErrorCode: the error code returned by OSS.
- ErrorMessage: the error message returned by OSS.
- RequestId: the UUID that identifies the request. If the issue persists, contact technical support personnel and provide this ID.

References

- Get started with Object Storage Service
- OSS domain names
- Error handling
- OSS error response

9.Use the COPY or UNLOAD statement to import or export data between OSS foreign tables and local tables

This topic describes how to use the COPY or UNLOAD statement to import or export data between Object Storage Service (OSS) foreign tables and local tables. allows you to use the COPY statement to import data from OSS foreign tables to local tables. You can also use the UNLOAD statement to export data from local tables to OSS foreign tables.

The COPY or UNLOAD statement is used to import or export data based on OSS foreign tables. For more information, see Use OSS foreign tables to access OSS data.

Precautions

When you use the COPY or UNLOAD statement to export data and store the data in a CSV file, you must enclose some options in quotation marks (") and write the options in lowercase letters. If you do not follow this requirement, some options may be the same as keywords. This may result in syntax errors. You must specify the following options in a particular way: `delimiter` , `quote` , `null` , `header` , `escape` , and `encoding` . Example:

```
UNLOAD ('select * from table') TO 'path' ACCESS_KEY_ID 'id' SECRET_ACCESS_KEY 'key' FORMAT
csv "delimiter" '|' "quote" '"' "null" '' "header" 'true' "escape" 'E' "encoding" 'utf-8' F
DW 'oss_fdw' ENDPOINT 'endpoint';
```

COPY

Syntax

```
COPY table-name [ column-list ] FROM data_source ACCESS_KEY_ID '<access-key-id>' SECRET
_ACCESS_KEY '<secret-access-key>' [ [ FORMAT ] [ AS ] data_format ] [ MANIFEST ] [ opti
on 'value' [ ... ] ]
```

- table_name: the name of the local table in which the imported data is stored. The local table must exist in the AnalyticDB for PostgreSQL instance.
- column-list: optional. The list of columns to which you want to write data. If you do not specify this parameter, data is written to all columns.
- data_source: the URL of the OSS bucket from which data is obtained. Example: *oss://bucket_name/path_prefix*.
- <access-key-id>: the AccessKey ID of the OSS account.
- <secret-access-key>: the AccessKey secret of the OSS account.
- [FORMAT] [AS] data_format: optional. The file format in which the imported data is stored. By default, FORMAT AS CSV is used. You can set data_format to BINARY, CSV, JSON, JSONLINE, ORC, PARQUET, or TEXT. In the parameter name, FORMAT and AS can be omitted. For example, FORMAT AS CSV or FORMAT CSV is equivalent to CSV.
- [MANIFEST]: specifies that the data source is a manifest file. The manifest file must be in the

JSON format and consist of the following elements:

- entries : an array of the OSS objects in the manifest file. The OSS objects can be in different buckets or paths. The OSS objects must be accessible by using the same ACCESS_KEY_ID or SECRET_ACCESS_KEY. Example:

```
{   "entries": [      {"url": "oss://adbpg-regress/local_t/_seg2_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress/local_t/_seg1_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress/local_t/_seg0_0.csv", "mandatory":true},       {"url": "oss://adbpg-regress-2/local_t/_seg1_0.csv", "mandatory":true},       {"url": "oss://adbpg-regress-2/local_t/_seg2_0.csv", "mandatory":true},       {"url": "oss://adbpg-regress-2/local_t/_seg0_0.csv", "mandatory":true}   ] }
```

- url : the full path of an OSS object.
- mandatory : specifies whether to report an error when an OSS object is not found.

- [option [value] [...]]: a list of one or more options. Specify each option in the key-value pair format. The following table describes the available options.

Option	Type	Required	Description
endpoint	String	Yes	The endpoint of the OSS bucket.
fdw	String	Yes	The name of the oss_fdw plug-in. The oss_fdw plug-in is required when you create a temporary OSS server for the COPY statement.
Other options that are used to create an OSS foreign table, including format, filetype, delimiter, and escape	N/A	N/A	The options that are used to create a temporary OSS foreign table. For more information, see Use OSS foreign tables to access OSS data.

Example 1

1. Create a local table.

```
postgres=# create table local_t2 (a int, b float8, c text);
```

2. Use the COPY statement to import data to columns a and c. Column b is assigned NULL.

```
postgres=# COPY local_t2 (a, c) FROM 'oss://adbpg-regress/local_t/' ACCESS_KEY_ID 'id' SECRET_ACCESS_KEY 'key' FORMAT AS CSV ENDPOINT '<endpoint>' FDW 'oss_fdw'; postgres=# select * from local_t2 limit 10; a | b | c ----+---+---------------------------------- 12 | | a24cba6ebdc5e0c485cd88ef60b72fea 15 | | c4d3028f5205fab98e5f43c7945db4ba 20 | | 769884311db01f400e21a903a3f1cb50 26 | | 7d12c981d262e0067ea1a04368f32f2a 30 | | 4e64bda52d54d263d16f42771b1d0225 35 | | b70c976d4c04568bd497b42a7d2e451d 40 | | d07ce2948b8618b47c351b6e222182f6 46 | | c2234393f878f5557776b7e778299564 47 | | cde904b2331fa274cd8d9266aa858342 50 | | 1235b900fb644bb36440a274314e4b6b (10 rows)
```

3. Check whether the data in columns a and c of the local_t2 table is the same as that of the local_t table.

```
postgres=# select sum(hashtext(t.a::text)) as col_a_hash, sum(hashtext(t.c::text)) as col_c_hash from local_t2 t; col_a_hash | col_c_hash -------------+------------- 23725368368 | 13447976580 (1 row) postgres=# select sum(hashtext(t.a::text)) as col_a_hash, sum(hashtext(t.c::text)) as col_c_hash from local_t t; col_a_hash | col_c_hash -------------+------------- 23725368368 | 13447976580 (1 row)
```

4. Save the data in a format other than CSV.

```
-- Save the data in the ORC format. COPY tt FROM 'oss://adbpg-regress/q_oss_orc_list/' ACCESS_KEY_ID 'id' SECRET_ACCESS_KEY 'key' FORMAT AS ORC ENDPOINT '<endpoint>' FDW 'oss_fdw'; -- Save the data in the PARQUET format. COPY tp FROM 'oss://adbpg-regress/test_parquet/' ACCESS_KEY_ID 'id' SECRET_ACCESS_KEY 'key' FORMAT AS PARQUET ENDPOINT '<endpoint>' FDW 'oss_fdw';
```

Example 2

1. Create a local table.

```
 create table local_manifest (a int, c text);
```

2. Create a manifest file, in which the OSS objects can be in different buckets.

```
{ "entries": [ {"url": "oss://adbpg-regress/local_t/_20210114103840_83f407434beccbd4eb2a0ce45ef39568_1450404435_seg2_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress/local_t/_20210114103840_83f407434beccbd4eb2a0ce45ef39568_1856683967_seg1_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress/local_t/_20210114103840_83f407434beccbd4eb2a0ce45ef39568_1880804901_seg0_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress-2/local_t/_20210114103849_67100080728ef95228e662bc02cb99d1_1008521914_seg1_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress-2/local_t/_20210114103849_67100080728ef95228e662bc02cb99d1_1234881553_seg2_0.csv", "mandatory":true}, {"url": "oss://adbpg-regress-2/local_t/_20210114103849_67100080728ef95228e662bc02cb99d1_1711667760_seg0_0.csv", "mandatory":true} ] }
```

3. Use the COPY statement to import the data from the manifest file to the local table.

```
-- Import an OSS object from the manifest file. COPY local_manifest FROM 'oss://adbpg-regress-2/unload_manifest/t_manifest' ACCESS_KEY_ID '<id>' SECRET_ACCESS_KEY '<key>' FORMAT AS CSV MANIFEST -- Specifies that the data source is a manifest file. ENDPOINT '<endpoint>' FDW 'oss_fdw';
```

Example 3

When you use the COPY statement to import data from OSS, error lines may be returned. In this case, you can set the following options to implement fault tolerance:

- log_errors: specifies whether to record information of the error lines in log files.
- segment_reject_limit: Up to 10 error lines are allowed. If the number of error lines exceeds 10, the system returns an error and exits.

1. Create a local table.

```
create table sales(id integer, value float8, x text) distributed by (id);
```

2. Use the COPY statement to import data from an OSS object that has three error lines.

```
COPY sales FROM 'oss://adbpg-const/error_sales/' ACCESS_KEY_ID '<id>' SECRET_ACCESS
_KEY '<key>' FORMAT AS csv log_errors 'true' -- The information of the error lines
is recorded in log files. segment_reject_limit '10' -- Up to 10 error lines are all
owed. If the number of error lines exceeds 10, the system returns an error and exit
s. endpoint '<endpoint>' FDW 'oss_fdw'; NOTICE: found 3 data formatting errors (3 o
r more input rows), rejected related input data COPY FOREIGN TABLE
```

3. Execute the following statement to query the error line details:

```
select * from gp_read_error_log('<Name of the source table on which the COPY statem
ent is executed>');
```

In the following example, the error line details of the sales table are queried:

```
select * from gp_read_error_log('sales'); cmdtime | relname | filename | linenum |
bytenum | errmsg | rawdata | rawbytes -------------------------------+------------
-----------------------------------+-------------------------+---------+---------+-
-------------------------------------------------------+---------+---------- 202
1-02-08 14:24:04.225238+08 | adbpgforeigntabletmp_20210208142403_1936866966 | error
_sales/sales.2.csv | 2 | | invalid byte sequence for encoding "UTF8": 0xed 0xab 0xa
d | | \x 2021-02-08 14:24:04.225238+08 | adbpgforeigntabletmp_20210208142403_193686
6966 | error_sales/sales.2.csv | 3 | | invalid byte sequence for encoding "UTF8": 0
xed 0xab 0xad | | \x 2021-02-08 14:24:04.225269+08 | adbpgforeigntabletmp_202102081
42403_1936866966 | error_sales/sales.3.csv | 2 | | invalid byte sequence for encodi
ng "UTF8": 0xed 0xab 0xad | | \x (3 rows)
```

Note The additionally saved error line logs occupy storage space. You can use the following syntax to delete the error line logs: `select gp_truncate_error_log('<Name of the table>')`.

UNLOAD

Syntax

```
UNLOAD ('select-statement') TO destination_url ACCESS_KEY_ID '<access-key-id>' SECRET_A
CCESS_KEY '<secret-access-key>' [ [ FORMAT ] [ AS ] data_format ] [ MANIFEST [ '<manife
st_url>' ] ] [ PARALLEL [ { ON | TRUE } | { OFF | FALSE } ] ] [ option 'value' [ ... ]
]
```

The following section describes the parameters:

- select-statement: the SELECT query statement. The query result data is written to OSS.

- destination_url: the URL of the OSS bucket in which the query result data is stored. Example: *oss://bucket-name/path-prefix*.
- <access-key-id>: the AccessKey ID of the OSS account.
- <secret-access-key>: the AccessKey secret of the OSS account.
- [FORMAT] [AS] data_format: optional. The file format in which the exported data is stored. By default, FORMAT AS CSV is used. You can set data_format to CSV, ORC, or TEXT. In the parameter name, FORMAT and AS can be omitted. For example, FORMAT AS CSV or FORMAT CSV is equivalent to CSV.
- MANIFEST: specifies to generate a manifest file when data is imported. If you specify <manifest_url> , the full path of the manifest file is different from that of the data files that are located in a different bucket. The path must end with the manifest suffix. If you do not specify <manifest_url> , the path of the manifest file is the same as that of the data files.

  Note If the file that is specified by <manifest_url> already exists, you must set the allowoverwrite option to true in [option [value] [...]] to overwrite the manifest file.

- PARALLEL: specifies whether to export data from multiple compute nodes in parallel. By default, data is exported from multiple compute nodes in parallel. A separate exported file is generated for each compute node. If you set this parameter to OFF/FALSE , the data is not exported in parallel. If the size of the exported data does not exceed 8 GB, the exported data is saved in a single file.
- [option [value] [...]]: a list of one or more options. Specify each option in the key-value pair format. The following table describes the available options.

Option	Type	Required	Description
endpoint	String	Yes	The endpoint of the OSS bucket.
fdw	String	Yes	The name of the oss_fdw plug-in. The oss_fdw plug-in is required when you create a temporary OSS server for the UNLOAD statement.
allowoverwrite	Boolean	No	Specifies whether to overwrite the existing manifest file. **Note** Only the manifest file can be overwritten. The data files cannot be overwritten.

Option	Type	Required	Description
Other options that are used to create an OSS foreign table, including format, filetype, delimiter, and escape	N/A	N/A	The options that are used to create a temporary OSS foreign table. For more information, see Use OSS foreign tables to access OSS data.

Example 1

1. Create a local table and insert test data into the table.

```
postgres=# create table local_t (a int, b float8, c text); postgres=# insert into l
ocal_t select r, random() * 1000, md5(random()::text) from generate_series(1,1000)r
; INSERT 0 1000 postgres=# select * from local_t limit 5; a | b | c ----+----------
--------+---------------------------------- 5 | 550.81393988803 | 8009fa725372e9967
86849213a695ce0 6 | 95.8335199393332 | ce7952c6728cdffdee06cc5b502d6457 9 | 421.379
795763642 | d3260ccbf6b9c03f3658d96bb7678b4d 10 | 362.347379792482 | 2bbbf89d23a2f8
3b089b589f55b5c4fc 11 | 800.203878898174 | a52994c5573e6b36d8a1c357bf800ce5 (5 rows
)
```

2. Use the UNLOAD statement to export the data from the specified columns of the local table to OSS and save the data in the CSV format.

```
postgres=# UNLOAD ('select a, c from local_t') TO 'oss://adbpg-regress/local_t/' AC
CESS_KEY_ID '<id>' SECRET_ACCESS_KEY '<key>' FORMAT AS CSV ENDPOINT '<endpoint>' FD
W 'oss_fdw'; NOTICE: OSS output prefix: "local_t/adbpgforeigntabletmp_2020090716480
1_1354519958_20200907164801_652261618". UNLOAD
```

3. Check whether the CSV file is written to the specified path.

```
$ ossutil --config hangzhou-zmf.config ls oss://adbpg-regress/local_t/ LastModified
Time Size(B) StorageClass ETAG ObjectName 2020-09-07 16:48:01 +0800 CST 12023 Stand
ard 9F38B5407142C044C1F3555F00000000 oss://adbpg-regress/local_t/adbpgforeigntablet
mp_20200907164801_1354519958_20200907164801_652261618_seg0_0.csv 2020-09-07 16:48:0
1 +0800 CST 12469 Standard 807BA680A0DED49BC1F3555F00000000 oss://adbpg-regress/loc
al_t/adbpgforeigntabletmp_20200907164801_1354519958_20200907164801_652261618_seg1_0
.csv 2020-09-07 16:48:01 +0800 CST 12401 Standard 3524F68F628CEB64C1F3555F00000000
oss://adbpg-regress/local_t/adbpgforeigntabletmp_20200907164801_1354519958_20200907
164801_652261618_seg2_0.csv Object Number is: 3 0.153414(s) elapsed
```

   Check whether the CSV file contains only the data in columns a and c of the local_t table.

```
$ head -n 10 adbpgforeigntabletmp_20200907164801_1354519958_20200907164801_65226161
8_seg2_0.csv 7,1225341d0d367a69b1b345536b21ef73 19,424a7a5c36066842f4de8c8a8341fc89
27,c214432e9928e4a6f7bef7bd815424c0 29,ade5d636e2b5d2a606a02e79255da4bd 37,85660e60
ede47b68493f6295620db568 77,e1be448ba2b08f0a2ca05b7ed812abfd 80,5e85d597a3b0f2f9736
a728724a0f9e0 92,dc23f76f0b1446504b8f1c2274521d2f 94,50304822488d55a500e3a71bcf4089
0f 97,e970fde8cd0df9c6b610925a488f6042
```

Example 2

1. Use the UNLOAD statement to export data and generate a manifest file. The path of the

manifest file is the same as that of the data files.

```
-- Use the UNLOAD statement to export data and generate a manifest file. UNLOAD ('s
elect * from local_t') TO 'oss://adbpg-regress/local_t/' ACCESS_KEY_ID '<id>' SECRE
T_ACCESS_KEY '<key>' FORMAT AS CSV MANIFEST -- Specifies to generate a manifest fil
e when the UNLOAD statement is used to export data. The path of the manifest file i
s the same as that of the data files. ENDPOINT '<endpoint>' FDW 'oss_fdw'; NOTICE:
OSS output prefix: "local_t/_20210114100329_3e9b07726306d88b3193dc95c10a5c5c". UNLO
AD -- View the list of exported files. The list includes several data files and a m
anifest file. ossutil ls -s oss://adbpg-regress/local_t/ oss://adbpg-regress/local_
t/_20210114100329_3e9b07726306d88b3193dc95c10a5c5c_162488956_seg1_0.csv oss://adbpg
-regress/local_t/_20210114100329_3e9b07726306d88b3193dc95c10a5c5c_163756258_seg0_0.
csv oss://adbpg-regress/local_t/_20210114100329_3e9b07726306d88b3193dc95c10a5c5c_17
41120517_seg2_0.csv oss://adbpg-regress/local_t/_20210114100329_3e9b07726306d88b319
3dc95c10a5c5c_manifest Object Number is: 4 0.136180(s) elapsed -- View the content
of the manifest file. ossutil cat oss://adbpg-regress/local_t/_20210114100329_3e9b0
7726306d88b3193dc95c10a5c5c_manifest { "entries": [ {"url": "oss://adbpg-regress/lo
cal_t/_20210114100329_3e9b07726306d88b3193dc95c10a5c5c_162488956_seg1_0.csv"}, {"ur
l": "oss://adbpg-regress/local_t/_20210114100329_3e9b07726306d88b3193dc95c10a5c5c_1
63756258_seg0_0.csv"}, {"url": "oss://adbpg-regress/local_t/_20210114100329_3e9b077
26306d88b3193dc95c10a5c5c_1741120517_seg2_0.csv"} ] }
```

2. Use the UNLOAD statement to export data and generate a specified manifest file. The path of the manifest file may be different from that of the data files.

Note If you set the ALLOWOVERWRITE option to true, the existing manifest file is overwritten. However, the data files are not overwritten. The data files can be manually deleted.

```
-- Use the UNLOAD statement to export data and generate a specified manifest file.
The manifest file is stored in a bucket that is different from that of the data fil
es. UNLOAD ('select * from local_t') TO 'oss://adbpg-regress/local_t/' ACCESS_KEY_I
D '<id>' SECRET_ACCESS_KEY '<key>' FORMAT AS CSV MANIFEST 'oss://adbpg-regress-2/un
load_manifest/t_manifest' -- Use the UNLOAD statement to export data and generate a
specified manifest file. ALLOWOVERWRITE 'true' -- Overwrite the existing manifest f
ile. ENDPOINT '<endpoint>' FDW 'oss_fdw'; NOTICE: OSS output prefix: "local_t/_2021
0114100329_3e9b07726306d88b3193dc95c10a5c5c". UNLOAD -- View the list of exported f
iles. Only data files are included in the list. ossutil ls -s oss://adbpg-regress/l
ocal_t/ oss://adbpg-regress/local_t/_20210114100956_4d3395a9501f6e22da724a2b6df1b6d
3_1736161168_seg0_0.csv oss://adbpg-regress/local_t/_20210114100956_4d3395a9501f6e2
2da724a2b6df1b6d3_1925769064_seg2_0.csv oss://adbpg-regress/local_t/_20210114100956
_4d3395a9501f6e22da724a2b6df1b6d3_644328153_seg1_0.csv Object Number is: 3 0.118540
(s) elapsed -- View the content of the manifest file that is stored in a bucket tha
t is different from that of the data files. ossutil cat oss://adbpg-regress-2/unloa
d_manifest/t_manifest { "entries": [ {"url": "oss://adbpg-regress/local_t/_20210114
100956_4d3395a9501f6e22da724a2b6df1b6d3_1736161168_seg0_0.csv"}, {"url": "oss://adb
pg-regress/local_t/_20210114100956_4d3395a9501f6e22da724a2b6df1b6d3_1925769064_seg2
_0.csv"}, {"url": "oss://adbpg-regress/local_t/_20210114100956_4d3395a9501f6e22da72
4a2b6df1b6d3_644328153_seg1_0.csv"} ] }
```

10.Use external tables for federated analytics of Hadoop data sources

AnalyticDB for PostgreSQL allows you to query external Hadoop data sources.

Note

- This feature is available only for AnalyticDB for PostgreSQL instances in elastic storage mode. The AnalyticDB for PostgreSQL instances must be in the same virtual private cloud (VPC) as the external data sources.
- This feature is unavailable for AnalyticDB for PostgreSQL instances in elastic storage mode that were created before September 6, 2020. This is because these instances cannot be connected to external Hadoop data sources over different network architectures. If you want to apply this feature to the existing instances, we recommend that you contact Alibaba Cloud technical support to apply for new instances and migrate data.

Configure a server

Server configurations vary based on user requirements. To query external Hadoop data sources for federated analytics, you must submit a ticket to request technical support to configure a server. The following table describes the information that is required when you submit a ticket.

External data source	Required information
Hadoop (HDFS, Hive, and HBase)	core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml, and hive-site.xml Note Profiles such as keytab and krb5.conf are also required for Kerberos authentication.

Syntax

Create an extension

```
CREATE extension pxf;
```

Create an external table

```
CREATE EXTERNAL TABLE <table_name> ( <column_name> <data_type> [, ...] | LIKE <other_ta
ble> ) LOCATION('pxf://<path-to-data>?PROFILE[&<custom-option>=<value>[...]]&[SERVER=va
lue]') FORMAT '[TEXT|CSV|CUSTOM]' (<formatting-properties>);
```

For more information about the syntax that is used to create an external table, see CREATE EXTERNAL TABLE.

Parameter	Description
path-to-data	The path to the data to be queried. Example: *//data/pxf_examples/pxf_hdfs_simple.txt*.
PROFILE [&<custom-option>=<value>[...]]	The profile that is used to query external data based on the Platform Extension Framework (PXF). Valid values: Jdbc, hdfs:text, hdfs:text:multi, hdfs:avro, hdfs:json, hdfs:parquet, hdfs:AvroSequenceFile, hdfs:SequenceFile, HiveText, HiveRC, HiveORC, HiveVectorizedORC, and HBase.
FORMAT '[TEXT|CSV|CUSTOM]'	The file format of the data to be queried. Valid values: TEXT, CSV, and CUSTOM.
formatting-properties	The formatting option of the specified file format. Set this parameter to formatter or delimiter. • Valid values when you set FORMAT to CUSTOM: ○ formatter='pxfwritable_import' ○ formatter='pxfwritable_export' • Valid values when you set FORMAT to TEXT or CSV: ○ delimiter=E'\t' ○ delimiter ':' **Note** E is used to escape possible special characters.
SERVER	The location of the profile on the server, which is provided by the technical support of AnalyticDB for PostgreSQL. • postgresql • hdp3

Query Hadoop Distributed File System (HDFS) data

The following table describes the supported data formats.

Data format	Profile
text	hdfs:text
csv	hdfs:text:multi and hdfs:text
Avro	hdfs:avro

Data format	Profile
JSON	hdfs:json
Parquet	hdfs:parquet
AvroSequenceFile	hdfs:AvroSequenceFile
SequenceFile	hdfs:SequenceFile

For more information about `FORMAT` and `formatting-properties`, see the "" section of this topic.

Example: Query an HDFS file

Create a test data file named `pxf_hdfs_simple.txt` on HDFS.

```
echo 'Prague,Jan,101,4875.33 Rome,Mar,87,1557.39 Bangalore,May,317,8936.99 Beijing,Jul,
411,11600.67' > /tmp/pxf_hdfs_simple.txt # Create a directory. hdfs dfs -mkdir -p /data
/pxf_examples # Put the file into the directory on HDFS. hdfs dfs -put /tmp/pxf_hdfs_si
mple.txt /data/pxf_examples/ # View the file. hdfs dfs -cat /data/pxf_examples/pxf_hdfs
_simple.txt
```

Create an external table and query it on an AnalyticDB for PostgreSQL instance.

```
postgres=# CREATE EXTERNAL TABLE pxf_hdfs_textsimple(location text, month text, num_ord
ers int, total_sales float8) LOCATION ('pxf://data/pxf_examples/pxf_hdfs_simple.txt?PRO
FILE=hdfs:text&SERVER=hdp3') FORMAT 'TEXT' (delimiter=E','); postgres=# select * from p
xf_hdfs_textsimple; location | month | num_orders | total_sales -----------+-------+---
---------+-------------------- Prague | Jan | 101 | 4875.3299999999999 Rome | Mar | 87
| 1557.3900000000001 Bangalore | May | 317 | 8936.9899999999998 Beijing | Jul | 411 | 1
1600.67 (4 rows)
```

The following list describes the parameters in the LOCATION clause:

- pxf:// : the pxf protocol. It cannot be modified.
- data/pxf_examples/pxf_hdfs_simple.txt : the *pxf_hdfs_simple.txt* file on HDFS.
- PROFILE=hdfs:text : the profile that is used to query HDFS data. In this example, *hdfs:text* is used.
- SERVER=hdp3 : the location of the profile used to query HDFS data based on PXF. In this example, *PXF_SERVER/hdp3/* is used. This value is provided by the technical support of AnalyticDB for PostgreSQL.
- FORMAT 'TEXT' (delimiter=E',') : the file format and formatting option of the external data source. In this example, the file format is set to `TEXT`, and the delimiter is set to commas (`,`).

For more information about parameter descriptions, see the "Syntax" section of this topic.

Example: Write data to HDFS in the TEXT or CSV format

The */data/pxf_examples/pxfwritable_hdfs_textsimple1* directory is created on HDFS by using the following command:

```
hdfs dfs -mkdir -p /data/pxf_examples/pxfwritable_hdfs_textsimple1
```

Note To execute INSERT statements on AnalyticDB for PostgreSQL, you must have write permissions on the preceding directory.

1. Create a writable external table on an AnalyticDB for PostgreSQL instance.

```
CREATE WRITABLE EXTERNAL TABLE pxf_hdfs_writabletbl_1(location text, month text, num_orders int, total_sales float8) LOCATION ('pxf://data/pxf_examples/pxfwritable_hdfs_textsimple1?PROFILE=hdfs:text&SERVER=hdp3') FORMAT 'TEXT' (delimiter=','); INSERT INTO pxf_hdfs_writabletbl_1 VALUES ( 'Frankfurt', 'Mar', 777, 3956.98 ); INSERT INTO pxf_hdfs_writabletbl_1 VALUES ( 'Cleveland', 'Oct', 3812, 96645.37 );
```

2. View information on HDFS.

```
# View the directory. hdfs dfs -ls /data/pxf_examples/pxfwritable_hdfs_textsimple1
# View data in the directory. hdfs dfs -cat /data/pxf_examples/pxfwritable_hdfs_textsimple1/* Frankfurt,Mar,777,3956.98 Cleveland,Oct,3812,96645.37
```

Query Hive data

Data format	Profile
TextFile	Hive and HiveText
SequenceFile	Hive
RCFile	Hive and HiveRC
Optimized Row Columnar (ORC)	Hive, HiveORC, and HiveVectorizedORC
Parquet	Hive

For more information about FORMAT and formatting-properties, see the "" section of this topic.

Example: Use the Hive profile

1. Generate data.

```
echo 'Prague,Jan,101,4875.33 Rome,Mar,87,1557.39 Bangalore,May,317,8936.99 Beijing,Jul,411,11600.67 San Francisco,Sept,156,6846.34 Paris,Nov,159,7134.56 San Francisco,Jan,113,5397.89 Prague,Dec,333,9894.77 Bangalore,Jul,271,8320.55 Beijing,Dec,100,4248.41' > /tmp/pxf_hive_datafile.txt
```

2. Create a Hive table.

```
hive> CREATE TABLE sales_info (location string, month string, number_of_orders int, total_sales double) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS textfile; hive> LOAD DATA LOCAL INPATH '/tmp/pxf_hive_datafile.txt' INTO TABLE sales_info; hive> SELECT * FROM sales_info;
```

3. Create an external table and query it on an AnalyticDB for PostgreSQL instance.

```
postgres=# create extension pxf; postgres=# CREATE EXTERNAL TABLE salesinfo_hivepro
file(location text, month text, num_orders int, total_sales float8) LOCATION ('pxf:
//default.sales_info?PROFILE=Hive&SERVER=hdp3') FORMAT 'custom' (formatter='pxfwrit
able_import'); postgres=# SELECT * FROM salesinfo_hiveprofile; location | month | n
um_orders | total_sales ---------------+-------+------------+------------- Prague |
Jan | 101 | 4875.33 Rome | Mar | 87 | 1557.39 Bangalore | May | 317 | 8936.99 Beiji
ng | Jul | 411 | 11600.67 San Francisco | Sept | 156 | 6846.34 Paris | Nov | 159 |
7134.56 ......
```

The following list describes the parameters in the LOCATION clause:

- pxf:// : the pxf protocol. It cannot be modified.
- default.sales_info : the *sales_info* table that is stored in the default database on Hive.
- PROFILE=Hive : the profile that is used to query Hive data. In this example, *Hive* is used.
- SERVER=hdp3 : the location of the profile used to query Hive data based on PXF. In this example, *PXF_SERVER/hdp3/* is used. This value is provided by the technical support of AnalyticDB for PostgreSQL.
- FORMAT 'custom' (formatter='pxfwritable_import') : the file format and formatting option of the external data source. In this example, the file format is set to *custom*, and the formatter is set to *pxfwritable_import*.

For more information about parameter descriptions, see the "Syntax" section of this topic.

Example: Use the HiveText profile

```
postgres=# CREATE EXTERNAL TABLE salesinfo_hivetextprofile(location text, month text, n
um_orders int, total_sales float8) LOCATION ('pxf://default.sales_info?PROFILE=HiveText
&SERVER=hdp3') FORMAT 'TEXT' (delimiter=E','); postgres=# select * from salesinfo_hivet
extprofile; location | month | num_orders | total_sales ---------------+-------+-------
-----+------------- Prague | Jan | 101 | 4875.33 Rome | Mar | 87 | 1557.39 Bangalore |
May | 317 | 8936.99 ......
```

Example: Use the HiveRC profile

1. Create a Hive table.

```
hive> CREATE TABLE sales_info_rcfile (location string, month string, number_of_orde
rs int, total_sales double) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS
rcfile; OK ## Import data. hive> INSERT INTO TABLE sales_info_rcfile SELECT * FROM
sales_info; ## View data. hive> SELECT * FROM sales_info_rcfile;
```

2. Create an external table and query it on an AnalyticDB for PostgreSQL instance.

```
postgres=# CREATE EXTERNAL TABLE salesinfo_hivercprofile(location text, month text,
num_orders int, total_sales float8) LOCATION ('pxf://default.sales_info_rcfile?PROF
ILE=HiveRC&SERVER=hdp3') FORMAT 'TEXT' (delimiter=E','); postgres=# SELECT location
, total_sales FROM salesinfo_hivercprofile; location | total_sales ---------------+
------------- Prague | 4875.33 Rome | 1557.39 Bangalore | 8936.99 ......
```

Example: Use the HiveORC profile

ORC-supporting profiles, HiveORC and HiveVectorizedORC, provide the following features:

- Read a single row of data at a time.
- Support column projection.

- Support complex types. You can query Hive tables that are composed of array, map, struct, and union data types.

  1. Create a Hive table.

     ```
     hive> CREATE TABLE sales_info_ORC (location string, month string, number_of_orders
     int, total_sales double) STORED AS ORC; hive> INSERT INTO TABLE sales_info_ORC SELE
     CT * FROM sales_info; hive> SELECT * FROM sales_info_ORC;
     ```

  2. Create an external table and query it on an AnalyticDB for PostgreSQL instance.

     ```
     postgres=# CREATE EXTERNAL TABLE salesinfo_hiveORCprofile(location text, month text
     , num_orders int, total_sales float8) LOCATION ('pxf://default.sales_info_ORC?PROFI
     LE=HiveORC&SERVER=hdp3') FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import'); postgres
     =# SELECT * FROM salesinfo_hiveORCprofile; ...... Prague | Dec | 333 | 9894.77 Bang
     alore | Jul | 271 | 8320.55 Beijing | Dec | 100 | 4248.41 (60 rows) Time: 420.920 m
     s
     ```

Example: Use the HiveVectorizedORC profile

This profile provides the following features:

- Reads up to 1,024 rows of data at a time.
- Does not support column projection.
- Does not support complex types or the timestamp data type.

Create an external table and query it on an AnalyticDB for PostgreSQL instance.

```
CREATE EXTERNAL TABLE salesinfo_hiveVectORC(location text, month text, num_orders int,
total_sales float8) LOCATION ('pxf://default.sales_info_ORC?PROFILE=HiveVectorizedORC&S
ERVER=hdp3') FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import'); select * from salesinfo_
hiveVectORC; location | month | num_orders | total_sales ---------------+-------+------
------+------------- Prague | Jan | 101 | 4875.33 Rome | Mar | 87 | 1557.39 Bangalore |
May | 317 | 8936.99 Beijing | Jul | 411 | 11600.67 San Francisco | Sept | 156 | 6846.34
......
```

Example: Query a Parquet-format Hive table

1. Create a Hive table.

   ```
   hive> CREATE TABLE hive_parquet_table (location string, month string, number_of_ord
   ers int, total_sales double) STORED AS parquet; INSERT INTO TABLE hive_parquet_tabl
   e SELECT * FROM sales_info; select * from hive_parquet_table;
   ```

2. Create an external table and query it on an AnalyticDB for PostgreSQL instance.

   ```
   postgres=# CREATE EXTERNAL TABLE pxf_parquet_table (location text, month text, numb
   er_of_orders int, total_sales double precision) LOCATION ('pxf://default.hive_parqu
   et_table?profile=Hive&SERVER=hdp3') FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import'
   ); postgres=# SELECT month, number_of_orders FROM pxf_parquet_table; month | number
   _of_orders -------+------------------ Jan | 101 Mar | 87 May | 317 Jul | 411 Sept |
   156 Nov | 159 ......
   ```

11.Use external tables for federated analytics of external SQL databases

AnalyticDB for PostgreSQL allows you to use Java Database Connectivity (JDBC) to query data from external data sources including Oracle, PostgreSQL, and MySQL databases.

Note

- This feature is available only for AnalyticDB for PostgreSQL instances in elastic storage mode. Moreover, the AnalyticDB for PostgreSQL instances must be in the same VPC as the external data sources.
- This feature is unavailable for the existing AnalyticDB for PostgreSQL instances in elastic storage mode that were created before September 6, 2020. This is because the existing instances cannot be connected to external databases over different network architectures. If you want to use this feature for the existing instances, we recommend that you contact Alibaba Cloud technical support to apply for new instances and migrate data.

Configure a JDBC server

JDBC server configurations vary based on user requirements. To use JDBC to query data from external data sources for federated analytics, you must submit a ticket to request technical support to configure the specified JDBC server. The following table describes the information that is required to query data from external data sources.

External data source	Required information
SQL Database(PostgreSQL, Mysql, Oracle)	The URL that the JDBC driver uses to connect to the database, the database username, and the database password.

Create an external table for federated analytics of an external SQL database

Create an extension

```
CREATE extension pxf;
```

Create an external table

```
CREATE EXTERNAL TABLE <table_name> ( <column_name> <data_type> [, ...] | LIKE <other_ta
ble> ) LOCATION('pxf://<path-to-data>? PROFILE[&<custom-option>=<value>[...]] &[SERVER=
value]') FORMAT '[TEXT|CSV|CUSTOM]' (<formatting-properties>);
```

For more information about the syntax that is used to create an external table, see CREATE EXTERNAL TABLE.

Field	Description
path-to-data	The path to the data to be queried. Example: *public.test_a*.
PROFILE [&<custom-option>=<value>[...]]	The method to query external data. To use JDBC to query data from external databases, set PROFILE to Jdbc.
FORMAT '[TEXT|CSV|CUSTOM]'	The format of the file to be queried.
formatting-properties	The formatting option of the specified data format. Set this field to formatter or delimiter. • Valid values when you set FORMAT to CUSTOM: ○ formatter='pxfwritable_import' ○ formatter='pxfwritable_export' • Valid values when you set FORMAT to TEXT or CSV: ○ delimiter=E'\t' ○ delimiter ':' **Note** E is used to escape possible special characters.
SERVER	The location of the configuration file of the server, which is provided by the technical support of AnalyticDB for PostgreSQL.

Example: Query data from an external PostgreSQL database

After the JDBC server is configured, you can execute the following SQL statements in AnalyticDB for PostgreSQL to create an external table and query data from the specified external PostgreSQL database:

```
postgres=# CREATE EXTERNAL TABLE pxf_ext_pg(a int, b int) LOCATION ('pxf://public.t? PROFIL
E=Jdbc&SERVER=postgresql') FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import') ENCODING 'UTF8'
; postgres=# select * from pxf_ext_pg; a | b -------+------- 1 | 2 2 | 4 3 | 6 4 | 8 5 | 10
6 | 12 7 | 14 --more--
```

The following list describes the fields in the LOCATION parameter:

- pxf:// : the PXF protocol, which cannot be modified.
- public.t : the t table in the public database to be queried.
- PROFILE=Jdbc : the method that is used to query external data. In this example, JDBC is used to query the external data source.
- SERVER=postgresql : the location of the configuration file of the server, which is provided by the technical support of AnalyticDB for PostgreSQL after you submit the ticket. In this example,

SERVER=postgresql indicates that the file located in the *PXF_SERVER/postgresql/* directory is used to configure the server.

- FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import') : the formatting option of the external data source. In this example, FORMAT is set to *CUSTOM* and FORMATTER is set to *pxfwritable_import*.

```
CREATE EXTERNAL TABLE pxf_ext_test_a( id int,name varchar) LOCATION ('pxf://public.test_a?
PROFILE=Jdbc&server=postgresql') FORMAT 'CUSTOM' (FORMATTER='pxfwritable_import') ENCODING
'UTF8';
```

Supported data types

- INTEGER, BIGINT, SMALLINT
- REAL, FLOAT8
- NUMERIC
- BOOLEAN
- VARCHAR, BPCHAR, TEXT
- DATE
- TIMESTAMP
- BYTEA

12.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.

13.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 (true, 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

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`
@>	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'`

Operator	Right operand type	Description	Example
?|	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]]`
`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`
`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::myrowtype, '{"a":1,"b":2}')`	`a | b ---+--- 1 | 2`

Function	Return value type	Description	Example	Return result example
`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_nam
e 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, n
um) values('{1,5}'::int[], '{"obj":1}', 5); INSERT 0 1 => select row_to_json(q) from (selec
t 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, o
bj, 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 loops=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 rows=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 max. 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. 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 rows=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 max. 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()
```

14.Use sort keys and rough set indexes to accelerate queries in column-oriented tables

This topic describes how to use sort keys and rough set indexes to improve query performance in column-oriented tables.

Notice This topic applies to the following instances:

- Newly created instances in reserved mode, whose kernel version is later than 20200826.
- Newly created instances in elastic mode, whose kernel version is later than 20200906.

Background information

When you create a table, you can define one or more columns as the sort key. After data is written to the table, you can sort the table data by sort key.

Sorting accelerates range-restricted queries. The minimum and maximum values for each column are stored in a database. If a query restricts the range in the WHERE clause, the query processor of AnalyticDB for PostgreSQL can use the minimum and maximum values to skip blocks out of the specific range during table scans.

For example, assume that a table stores seven years of data sorted by date, and a query specifies a date range of one month. In this case, only 1/(7 × 12) of the table data needs to be scanned, and 98.8% of the disk blocks can be eliminated from the scan. If the data is not sorted by date, all disk blocks may be scanned.

AnalyticDB for PostgreSQL supports the following sorting methods:

- Compound sorting: applies to scenarios where the confining condition (constraint in the WHERE clause) is a prefix subset of the sort key or consists of all columns of the sort key. This sorting method is more useful for scenarios where the query condition includes the primary column of the confining condition.
- Interleaved sorting: gives an equal weight to each column in the sort key. This sorting method is more useful for scenarios where the query condition includes a subset of the confining condition.

For more information, see the "Performance comparison between compound sorting and interleaved sorting" section of this topic.

Performance comparison

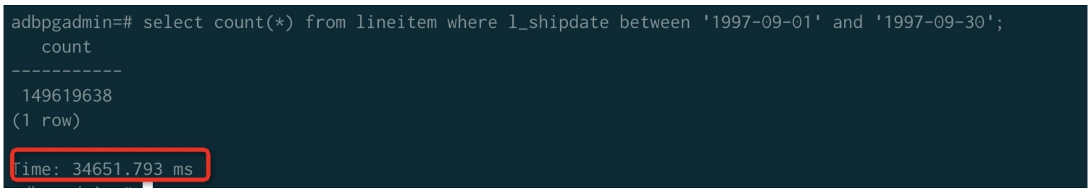
This section provides an example on how compound sorting improves query performance for rough set indexes when compared with a full table scan.

A TPC-H Lineitem table that stores seven years of data is used in this example. Then, queries on data that uses and does not use the l_shipdate field are compared. Both queries are range-restricted.

Note This implementation of TPC is derived from the TPC Benchmark and is not comparable to published TPC Benchmark results, as this implementation does not comply with all the requirements of the TPC Benchmark.

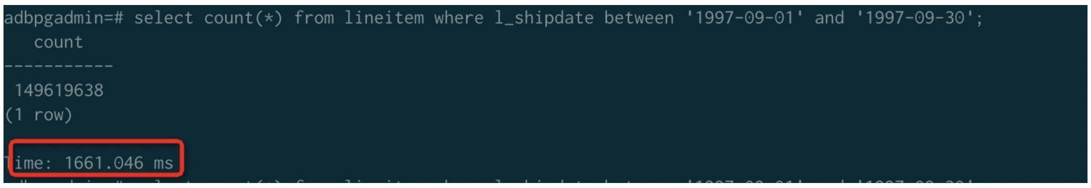
Test procedure:

1. Create a 32-node instance.
2. Write 13 billion rows to the Lineitem table.
3. Query data in a time range from 1997-09-01 to 1997-09-30.
   - Data is not sorted by l_shipdate.



```
adbpgadmin=# select count(*) from lineitem where l_shipdate between '1997-09-01' and '1997-09-30';
   count
-----------
 149619638
(1 row)

Time: 34651.793 ms
```

   - Data is sorted by l_shipdate.



```
adbpgadmin=# select count(*) from lineitem where l_shipdate between '1997-09-01' and '1997-09-30';
   count
-----------
 149619638
(1 row)

Time: 1661.046 ms
```

Define columns as the sort key when you create a table

Example

```
create table test(date text, time text, open float, high float, low float, volume int) with (APPENDONLY=true,ORIENTATION=column) ORDER BY (volume);
```

Syntax

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name ( [ { column_name data_type ...} ] ) [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ] [ ORDER BY (column, [ ... ] )]
```

If the kernel version is earlier than 20210326, use the following statement to specify the sort key: SORTKEY (column, [...])

Sort a table

Use compound sorting to sort data

```
SORT [tablename]
```

If the kernel version is earlier than 20210326, use the following statement to specify the sort key:

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

Use interleaved sorting to sort data

```
MULTISORT [tablename]
```

If the kernel version is earlier than 20210326, use the following statement to specify the sort key:

```
VACUUM REINDEX [tablename]
```

After you execute the `SORT` or `MULTISORT` statement on a table, the table is sorted based on the specified sort key. As you add rows to the sorted table, the amount of unsorted data increases and filtering performance in rough sets may degrade. To ensure the filtering performance in rough sets, you must execute the `SORT` or `VACUUM REINDEX` (MULTISORT) statement to resort the table on a regular basis.

Modify sort keys

You can execute the following statement to modify the sort key of an existing column-oriented table based on your business requirements:

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

This statement modifies only the catalog and does not immediately sort the data. To sort the data, you must execute the `SORT table_name` statement.

Example

```
ALTER TABLE test SET ORDER BY(high,low);
```

If the kernel version is earlier than 20210326, use the following statement to modify the sort key:

```
ALTER TABLE test SET SORTKEY(high,low);
```

Specify a sort key and choose a sorting method

If you always need to run SQL queries that contain one or more column values or that are range-restricted, such as date columns, you can use those columns as sort keys. This accelerates the preceding SQL queries by sorting data based on rough set indexes.

We recommend that you use compound sorting in general cases. Compound sorting needs to perform extra analysis on the data. Therefore, `VACUUM REINDEX` can take longer than `VACUUM SORT ONLY` for interleaved tables.

If you seldom need to run SQL queries that contain specific columns, you can use interleaved sorting to accelerate queries. An interleaved sort key can use up to eight columns.

Performance comparison between compound sorting and interleaved sorting

In this section, two tables that contain the same data are separately sorted by using compound sorting and interleaved sorting. The results of queries on the tables show that the two sorting methods show different performance levels in different scenarios.

Two tables with the same four columns (id, num1, num2, and value) are used in this example. (id,num1,num2) is specified as the sort key. Each table contains a total of 10 million rows. The tables used in this example are not particularly large tables for AnalyticDB for PostgreSQL, but the performance difference between compound sorting and interleaved sorting can be seen with the tables. The performance difference is more significant in large tables.

Test procedure:

1. Create two tables (test and test_multi) and set the same sort key for them.
2. Write test data to the tables.
3. Sort the test table by using compound sorting and sort the test_multi table by using interleaved sorting.
4. Compare the point query performance between compound sorting and interleaved sorting in the same SQL query.
5. Compare the range query performance between compound sorting and interleaved sorting in the same SQL query.

Create two tables and set the same sort key for them

```
CREATE TABLE test(id int, num1 int, num2 int, value varchar) with(APPENDONLY=TRUE, ORIENTATION=column) DISTRIBUTED BY(id) ORDER BY(id, num1, num2); CREATE TABLE test_multi(id int, num1 int, num2 int, value varchar) with(APPENDONLY=TRUE, ORIENTATION=column) DISTRIBUTED BY(id) ORDER BY(id, num1, num2);
```

Write 10 million rows of data to each table

```
INSERT INTO test(id, num1, num2, value) select g, (random()*10000000)::int, (random()*10000000)::int, (array['foo', 'bar', 'baz', 'quux', 'boy', 'girl', 'mouse', 'chlid', 'phone'])[floor(random() * 10 +1)] FROM generate_series(1, 10000000) as g; INSERT INTO test_multi SELECT * FROM test; adbpgadmin=# SELECT count(*) FROM test; count ---------- 10000000 (1 row) adbpgadmin=# SELECT count(*) FROM test_multi; count ---------- 10000000 (1 row)
```

Separately sort the two tables by using compound sorting and interleaved sorting

```
SORT test; MULTISORT test_multi;
```

Compare the point query performance

- The query is filtered on the primary column.

```
-- Q1 is filtered on the primary column. select * from test where id = 100000; select * from test_multi where id = 100000;
```

- The query is filtered on the second column.

```
-- Q2 is filtered on the second column. select * from test where num1 = 8766963; select * from test_multi where num1 = 8766963;
```

- The query is filtered on the second and third columns.

```
-- Q3 is filtered on the second and third columns. select * from test where num1 = 100000 and num2=2904114; select * from test_multi where num1 = 100000 and num2=2904114;
```

Performance comparison results

Sorting method	Q1	Q2	Q3
Compound sorting	0.026s	3.95s	4.21s

Sorting method	Q1	Q2	Q3
Interleaved sorting	0.55s	0.42s	0.071s

Compare the range query performance

- The query is filtered on the primary column.

```
-- Q1 is filtered on the primary column. select count(*) from test where id>5000 and id < 100000; select count(*) from test_multi where id>5000 and id < 100000;
```

- The query is filtered on the second column.

```
-- Q2 is filtered on the second column. select count(*) from test where num1 >5000 and num1 <100000; select count(*) from test_multi where num1 >5000 and num1 <100000;
```

- The query is filtered on the second and third columns.

```
-- Q3 is filtered on the second and third columns. select count(*) from test where num1 >5000 and num1 <100000; and num2 < 100000; select count(*) from test_multi where num1 >5000 and num1 <100000 and num2 < 100000;
```

Performance comparison results

Sorting method	Q1	Q2	Q3
Compound sorting	0.07s	3.35s	3.64s
Interleaved sorting	0.44s	0.28s	0.047s

Test conclusion

- Q1 uses the primary column of the sort key to filter data. In this case, compound sorting has a shorter query response time than interleaved sorting.
- Q2 uses a non-primary key column of the sort key to filter data. In this case, compound sorting is ineffective and interleaved sorting has significantly better query performance.
- Q3 uses non-primary key columns of the sort key to filter data. In this case, interleaved sorting is faster and more effective than compound sorting. The more columns of interleaved sort keys used, the better the performance of the interleaved sorting query.

15.Use the ANALYZE statement 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 invalid or outdated, the query optimizer may generate an inefficient query plan. You can use the ANALYZE statement to collect statistics.

Execute the ANALYZE statement to collect statistics:

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

AUTO ANALYZE

Auto-analyze automatically executes the ANALYZE statement. Auto-analyze checks for tables that have a large number of INSERT, UPDATE, or DELETE operations. When necessary, auto-analyze executes the ANALYZE statement on the tables to collect their statistics after the operations. By default, when those operations are performed on more than 10% of rows in a table, auto-analyze executes the ANALYZE statement on the table.

If an AnalyticDB for PostgreSQL instance has multiple coordinator nodes, auto-analyze checks for change operations only on the primary coordinator node. Auto-analyze cannot be triggered if change operations are performed on secondary coordinator nodes.

Note Auto-analyze is available only for 20210527 and later minor versions. For more information about how to upgrade the minor version, see Upgrade the engine version.

Generate statistics by table or column

The execution of the ANALYZE statement without parameters updates statistics for all tables in a database. This can be a long-running process. You can specify a table or column to collect its statistics. When your data changes, we recommend that you execute the ANALYZE statement on tables with changed data.

It takes a long time to execute the ANALYZE statement on a large table. To reduce the amount of 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 partition table, you can find its name in the pg_partitions system catalog by executing the following statement and then execute the ANALYZE statement on partitions with changed data:

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

When to execute the ANALYZE statement

Execute the ANALYZE statement after the following operations:

- Data loading
- CREATE INDEX operations

- A large number of INSERT, UPDATE, and DELETE operations

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

16.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 b
y 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)
```

17.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 parame
ter 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|MEDI
UM|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.
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.

Parameter	Description
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 o
ne 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 s
tatement_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'; qu
eueid | rsqname | rsqcountlimit | rsqcountvalue | rsqcostlimit | rsqcostvalue | rsqmemoryli
mit | rsqmemoryvalue | 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'; queu
eid | rsqname | rsqcountlimit | rsqcountvalue | rsqcostlimit | rsqcostvalue | rsqmemory
limit | rsqmemoryvalue | 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 (PRIO
RITY=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

18.Use MetaScan to accelerate queries for column-oriented tables

supports the column store model to provide a higher data compression ratio and query performance. However, when AnalyticDB for PostgreSQL processes a query that may return just a small amount of data, it must still read the data of an entire column or create B-tree indexes. B-tree indexes also have potential issues. For example, severe data bloat may occur because indexes are not compressed. If large amounts of query results are returned, indexes may cause higher costs than sequential scans or even fail. To resolve these issues, provides the MetaScan feature that offers excellent filter performance and occupies minimal storage space.

Notice

- The MetaScan feature is supported only for AnalyticDB for PostgreSQL instances in reserved storage mode that are created with the minor version of 20200826 or later.
- This feature is not supported for AnalyticDB for PostgreSQL instances in elastic storage mode.

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. When the value is ON, 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 value, submit 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 and change its value 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.

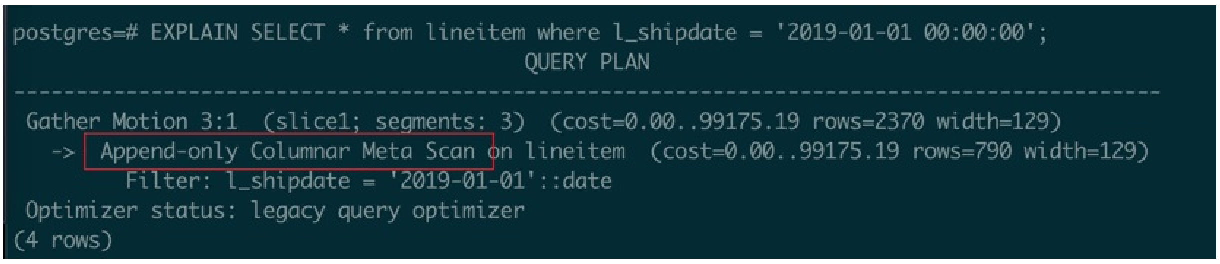
    ```
    SET RDS_ENABLE_COLUMN_META_SCAN = ON;
    ```

Notice When RDS_ENABLE_CS_ENHANCEMENT is set to OFF, metadata collection is disabled for all column-oriented tables. If you want to use the MetaScan feature when RDS_ENABLE_CS_ENHANCEMENT is set to ON, execute the following statement to re-collect the metadata of tables:

```
ALTER TABLE table_name SET WITH (REORGANIZE=TRUE);
```

Check whether MetaScan is used for a query

You can use the EXPLAIN statement to check whether a SELECT statement uses MetaScan, as shown in the following figure.



```
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 in the result is the MetaScan node. This indicates that the MetaScan feature is used for the query.

Data types and operators supported by MetaScan

MetaScan supports the following data types:

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

MetaScan supports the following operators:

- =, <, <=, >, and >=
- The AND logical operator

Update existing instances to use MetaScan

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

1. Update the minor version of your instance

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

2. Update the metadata of tables

   Update the metadata of tables to its latest version after you update the minor version. If RDS_ENABLE_CS_ENHANCEMENT is set to OFF, submit a ticket. If RDS_ENABLE_CS_ENHANCEMENT is set to ON, perform the following operations:

i. Create an update function as an administrator.

```
CREATE OR REPLACE FUNCTION UPGRADE_AOCS_TABLE_META(tname TEXT) RETURNS BOOL AS $$ D
ECLARE tcount INT := 0; BEGIN -- CHECK TABLE NAME EXECUTE 'SELECT COUNT(1) FROM PG_
AOCSMETA WHERE RELID = ''' || tname || '''::REGCLASS' INTO tcount; IF tcount IS NOT
NULL THEN IF tcount > 1 THEN RAISE EXCEPTION 'found more than one table of name %',
tname; ELSEIF tcount = 0 THEN RAISE EXCEPTION 'not found target table in pg_aocsmet
a, table name:%', tname; END IF; END IF; EXECUTE 'ALTER TABLE ' || tname || ' SET W
ITH(REORGANIZE=TRUE)'; RETURN TRUE; END; $$ LANGUAGE PLPGSQL;
```

ii. Execute the following SQL statement to update the metadata of tables as an administrator or a 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'::RE
GCLASS
```

Improve MetaScan performance by using sort keys

The sort key feature of sorts data in tables by a specific column. A combination of MetaScan with the sort key feature improves the performance of MetaScan. Column-oriented tables store data in blocks. MetaScan uses metadata to check whether blocks meet query conditions and skips blocks that do not meet query conditions. This reduces I/O operations and improves scanning performance. If data in filtered columns is distributed across all blocks, all blocks must be scanned even though a query may return a small amount of data. If you create a sort key for each filtered column, the same data in a column is combined into several consecutive blocks. This way, MetaScan can improve scanning performance by filtering out blocks that do not meet the query conditions.

For more information about how to create a sort key, see Use sort keys and rough set indexes to accelerate queries in column-oriented tables.

Limits

MetaScan is unavailable when 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. For more information about the ORCA optimizer, see Choose a query optimizer.

19.Configure sorting acceleration

This topic describes how to accelerate queries by sorting underlying data in .

After you execute a `SORT <tablename>` statement, the system sorts the data of the specified table. Then, pushes operators such as SORT down to the storage layer so that queries are accelerated based on the physical order of data. When your underlying data is ordered, your queries can be accelerated. This feature can accelerate queries that contain SORT, AGG, and JOIN operators based on sort keys.

Note

- The sorting acceleration feature requires all of your data to be ordered. After you write data, you must execute the `SORT <tablename>` statement again to order data.
- By default, the sorting acceleration feature is enabled.

Example

In this example, a test table named far is used to compare the query time before and after sorting acceleration.

1. Execute the following statement to create a test table named far:

```
CREATE TABLE far(a int,  b int)
WITH (APPENDONLY=TRUE, COMPRESSTYPE=ZSTD, COMPRESSLEVEL=5)
DISTRIBUTED BY (a)  --Distribution key
ORDER BY (a);       --Sort key
```

2. Execute the following statement to write 1,000,000 rows of data to the far table:

```
INSERT INTO far VALUES(generate_series(0, 1000000), 1);
```

3. Execute the following statement to sort the data in the far table:

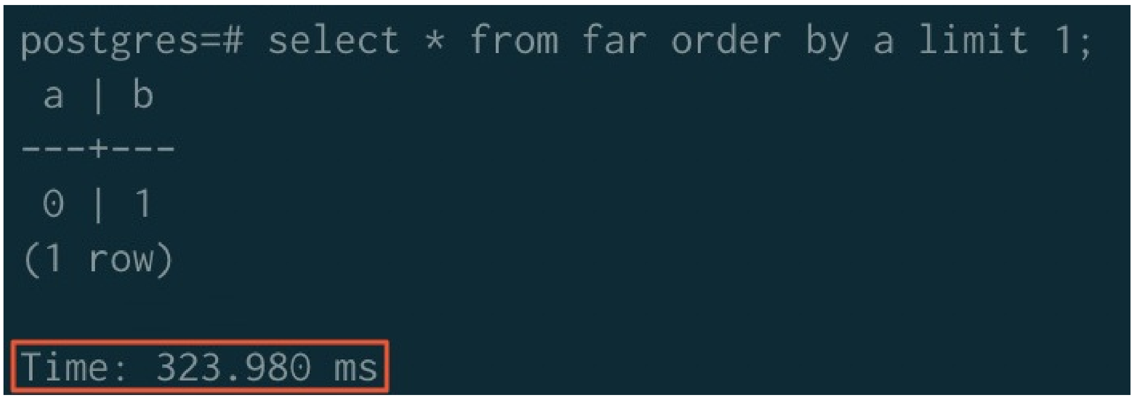
```
SORT far;
```

Query performance comparison:

Note The query time results in this example are for reference only. The query time varies based on various factors such as the data volume, computing resources, and network conditions.

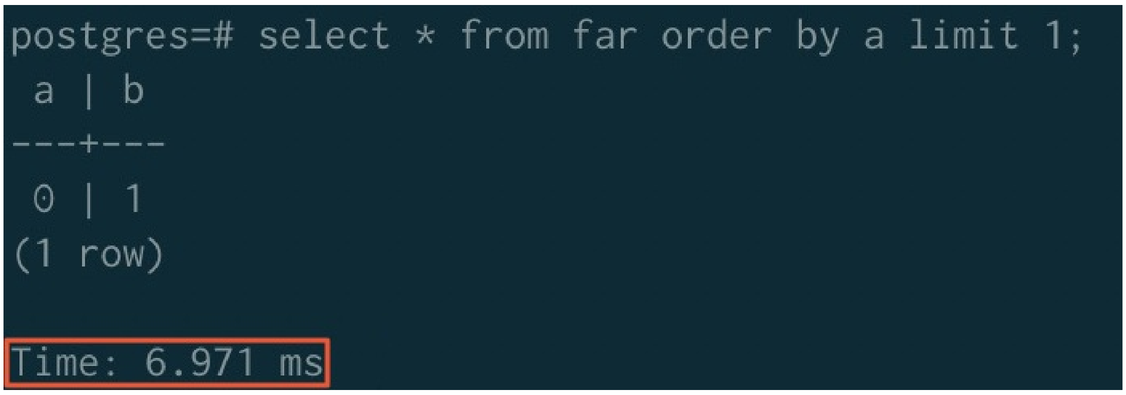
- ORDER BY sorting acceleration

- Before sorting acceleration



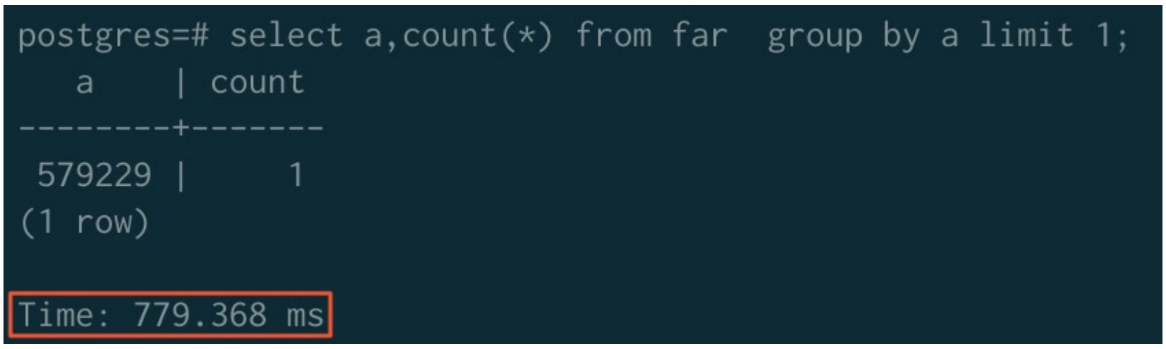
```
postgres=# select * from far order by a limit 1;
 a | b
---+---
 0 | 1
(1 row)

Time: 323.980 ms
```

- After sorting acceleration



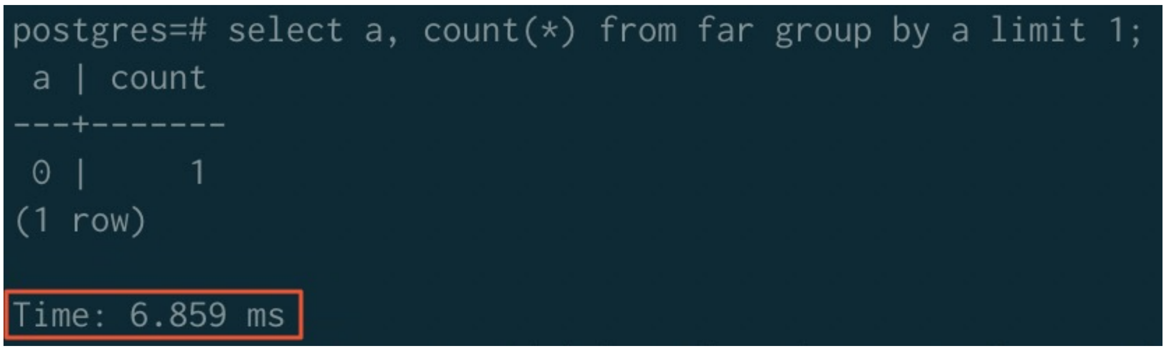
```
postgres=# select * from far order by a limit 1;
 a | b
---+---
 0 | 1
(1 row)

Time: 6.971 ms
```

- GROUP BY sorting acceleration
  - Before sorting acceleration



```
postgres=# select a,count(*) from far  group by a limit 1;
   a     | count
--------+-------
 579229 |     1
(1 row)

Time: 779.368 ms
```

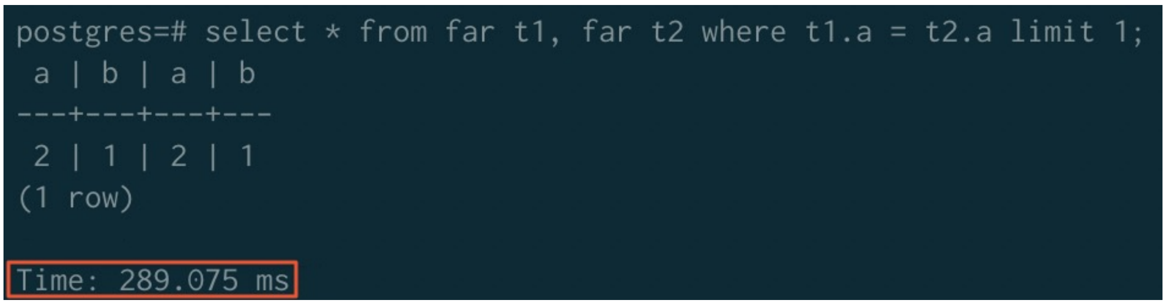
  - After sorting acceleration



```
postgres=# select a, count(*) from far group by a limit 1;
 a | count
---+-------
 0 |     1
(1 row)

Time: 6.859 ms
```

- JOIN sorting acceleration

- Before sorting acceleration

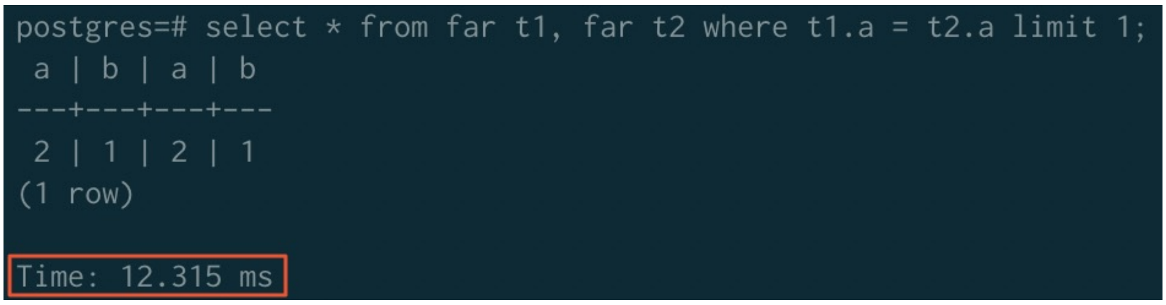


```
postgres=# select * from far t1, far t2 where t1.a = t2.a limit 1;
 a | b | a | b
---+---+---+---
 2 | 1 | 2 | 1
(1 row)

Time: 289.075 ms
```

- After sorting acceleration

Note If you want to use the sorting acceleration feature for JOIN operators, you must execute the following statements to disable the ORCA optimizer and enable the merge join algorithm:

```
SET enable_mergejoin TO on;
SET optimizer TO off;
```



```
postgres=# select * from far t1, far t2 where t1.a = t2.a limit 1;
 a | b | a | b
---+---+---+---
 2 | 1 | 2 | 1
(1 row)

Time: 12.315 ms
```

-	ORDER BY	GROUP BY	JOIN
Before acceleration	323.980 ms	779.368 ms	289.075 ms
After acceleration	6.971 ms	6.859 ms	12.315 ms

20.Auto-merge

This topic describes the auto-merge feature provided by .

Overview

Auto-merge is a process that runs in the backend of to automatically merge and sort data. This feature checks the status of data in a table on a regular basis. It sorts new unordered data and merges the data with the existing ordered data.

By default, the auto-merge feature is enabled for all tables. You can manually disable this feature for tables that do not require automatic data merging and sorting. For more information, see Enable or disable auto-merge on individual tables.

To disable this feature for all your tables, 提交工单Submit a ticket.

Note Auto-merge improves query performance, but consumes resources, especially I/O resources, in the merging and sorting process.

Precautions

- Auto-merge supports only append-optimized (AO) tables that are configured with sort keys. You can execute the following statement to check whether a table is an AO table:

```
SELECT reloptions FROM pg_class WHERE relname = 'table_name';
```

  The following results may be returned. `appendonly=true` indicates that the queried table is an AO table.

```
 reloptions ---------------------------------------------------------------------- {appen
donly=true,orientation=column,compresstype=lz4,compresslevel=9} (1 row)
```

- The auto-vacuum feature must be enabled, because auto-merge is dependent on auto-vacuum. For more information about auto-vacuum, see Configure scheduled maintenance tasks to clear junk data.
- The minor version of the instance must be 6.3.5.0 or later. For information about how to view and update the minor version of an instance, see Query the minor engine version and Upgrade the engine version.

Define sort keys

Before you use the auto-merge feature for a table, you must define sort keys for the table by using DDL statements.

- The following example shows how to define sort keys when you create a table:

```
CREATE TABLE table_name( col_name type, ... ) WITH(appendonly = true, orientation = row/c
olumn) DISTRIBUTED BY(distributed_keys) ORDER BY(sort_keys) ;
```

- The following example shows how to add or modify sort keys for an existing table:

```
ALTER TABLE table_name SET ORDER BY(sort_keys);
```

Enable or disable auto-merge on individual tables

allows you to enable or disable auto-merge on individual tables.

- Enable auto-merge on a table.

  ```
  ALTER TABLE table_name SET (automerge = on);
  ```

- Disable auto-merge on a table.

  ```
  ALTER TABLE table_name SET (automerge = off);
  ```

- The following example shows how to check whether auto-merge is enabled on a table:

  ```
  SELECT relname, reloptions FROM pg_class WHERE relname = 'table_name';
  ```

  Sample query result:

  ```
   relname | reloptions ------------+--------------------------------------------------- t
  able_name | {appendonly=true,orientation=column,automerge=on}
  ```

  `automerge=on` indicates that auto-merge is enabled. If the status of this feature on the specified table is the same as that on all tables of the instance, the automerge parameter is not returned. In this case, you can execute the following statement to query the value of this parameter:

  ```
  SHOW automerge;
  ```

Test the effect of auto-merge

In the following example, queries are performed to demonstrate the performance improvement brought by the auto-merge feature.

1. Check whether the auto-vacuum and auto-merge features are enabled.

   ```
   SHOW autovacuum; SHOW automerge;
   ```

   If both features are enabled, on is returned.

2. Disable the Laser computing engine.

   ```
   SET laser.enable = off;
   ```

   Note This operation is performed only to demonstrate the effect of the auto-merge feature. We recommend that you enable the Laser computing engine in real-case scenarios. For more information about the Laser computing engine, see Use the Laser computing engine.

3. Create a table.

   ```
   CREATE TABLE test_automerge(a bigint, b bigint) WITH(appendonly = true, orientation = c
   olumn, compresstype = lz4, compresslevel = 9) DISTRIBUTED BY(a) ORDER BY(b);
   ```

4. Specify the number of added rows that triggers the auto-merge feature and import random data. To compare the performance before and after auto-merge is triggered, make sure that the number of rows imported in this step is lower than the threshold.

   ```
   ALTER TABLE test_automerge set (automerge_unsorted_row_num_threshold = 10000000); INSER
   T INTO test_automerge SELECT i, round(random() * 100) FROM generate_series(1, 9000000)
   as i;
   ```

5. Perform a query.

```
SELECT count(*) FROM test_automerge WHERE b = 5;
```

Sample query result:

```
 count ------- 89713 (1 row) Time: 204.918 ms
```

6. Import more data to trigger auto-merge.

```
INSERT INTO test_automerge SELECT i, round(random() * 100) FROM generate_series(1, 1000
000) as i;
```

7. After the data is merged and sorted, perform a query again.

```
SELECT count(*) FROM test_automerge WHERE b = 5;
```

Sample query result:

```
 count ------- 99683 (1 row) Time: 15.289 ms
```

Note You can check the progress of auto-merge in the `pg_stat_activity` view.

The preceding example shows that auto-merge can significantly reduce the amount of time required for queries. In addition, auto-merge significantly improves performance in scenarios where the AGGREGATE, JOIN, or ORDER BY clauses are used. To maximize the effect of the auto-merge feature, we recommend that you select a sort key for a table based on the following rules:

- Specify one or more fields that filter out a large amount of data in queries as the sort key.
- If the AGGREGATE, JOIN, or ORDER BY clauses are frequently used on a table, select the group key, join key, or order key as the sort key. The distribution key and the sort key must be the same.

References

Configure sorting acceleration

21.Configure parallel query

provides parallel query for individual tables. This topic describes the parallel query feature.

Introduction

If your instance uses compute nodes that have 4 cores or higher specifications, parallel query is enabled by default for individual table queries to improve multi-core concurrency and reduce query time. The system automatically selects the degree of parallelism based on the number of concurrent queries, compute node specifications, and SQL statements. When the system detects a high workload, it disables parallel query. In low-concurrency scenarios, aggregate queries on individual tables that have large amounts of data can reduce the query time by about 50%.

Note

- Make sure that the minor version of your instance is 6.3.4.0 or later. For more information about how to update the minor version, see Upgrade the engine version.
- Parallel query can be used for instances whose compute nodes have 4 cores or higher specifications.

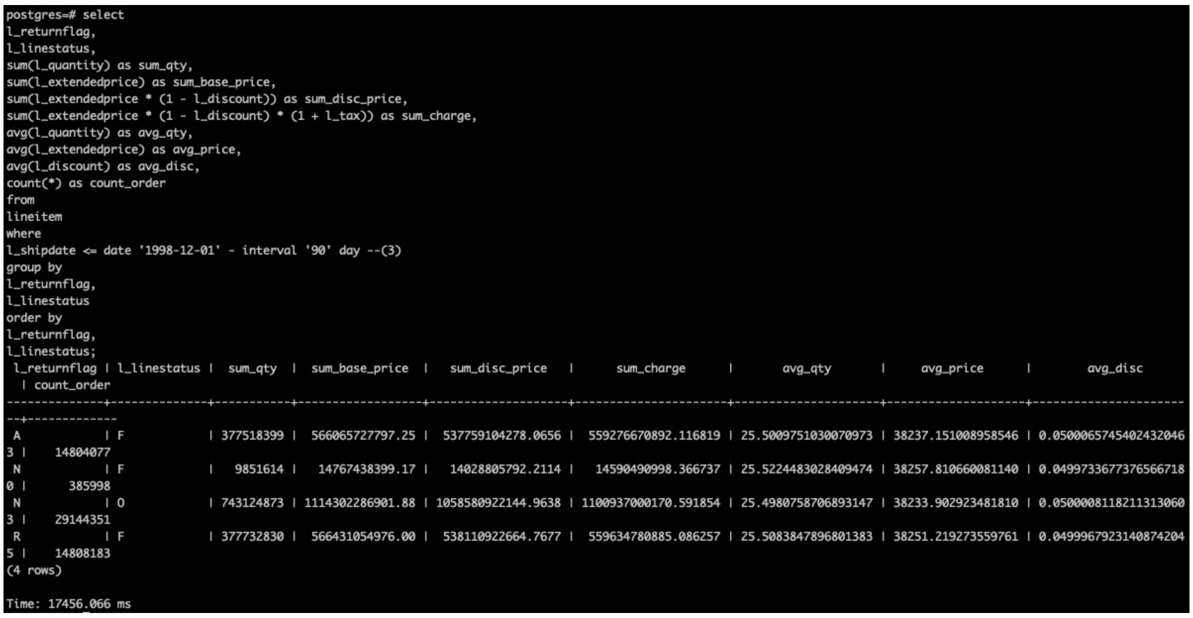
Example

In this example, an instance that has the following configurations is used:

- Compute node specifications: 4C32G
- Number of compute nodes: 4

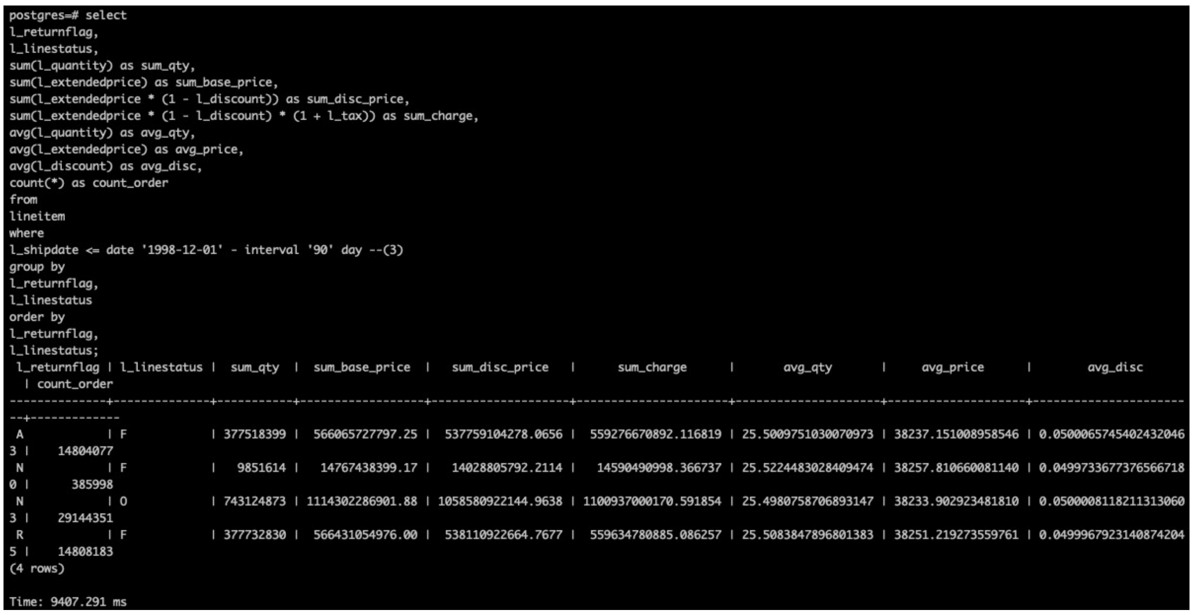
Query time of individual tables before and after parallel acceleration for 10 GB data:

- Before parallel acceleration



```
postgres=# select
l_returnflag,
l_linestatus,
sum(l_quantity) as sum_qty,
sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
avg(l_quantity) as avg_qty,
avg(l_extendedprice) as avg_price,
avg(l_discount) as avg_disc,
count(*) as count_order
from
lineitem
where
l_shipdate <= date '1998-12-01' - interval '90' day --(3)
group by
l_returnflag,
l_linestatus
order by
l_returnflag,
l_linestatus;
 l_returnflag | l_linestatus |  sum_qty  |  sum_base_price  |   sum_disc_price   |      sum_charge      |       avg_qty       |      avg_price      |       avg_disc
  | count_order
--------------+--------------+-----------+------------------+--------------------+----------------------+---------------------+---------------------+----------------------
--+-------------
 A            | F            | 377518399 |  566065727797.25 |  537759104278.0656 |  559276670892.116819 | 25.5009751030070973 | 38237.151008958546 | 0.0500065745402432046
3 |    14804077
 N            | F            |   9851614 |   14767438399.17 |   14028805792.2114 |   14590490998.366737 | 25.5224483028409474 | 38257.810660081140 | 0.0499733677376566718
0 |      385998
 N            | O            | 743124873 | 1114302286901.88 | 1058580922144.9638 | 1100937000170.591854 | 25.4980758706893147 | 38233.902923481810 | 0.0500008118211313060
3 |    29144351
 R            | F            | 377732830 |  566431054976.00 |  538110922664.7677 |  559634780885.086257 | 25.5083847896801383 | 38251.219273559761 | 0.0499967923140874204
5 |    14808183
(4 rows)

Time: 17456.066 ms
```

- After parallel acceleration



```
postgres=# select
l_returnflag,
l_linestatus,
sum(l_quantity) as sum_qty,
sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
avg(l_quantity) as avg_qty,
avg(l_extendedprice) as avg_price,
avg(l_discount) as avg_disc,
count(*) as count_order
from
lineitem
where
l_shipdate <= date '1998-12-01' - interval '90' day --(3)
group by
l_returnflag,
l_linestatus
order by
l_returnflag,
l_linestatus;
 l_returnflag | l_linestatus |  sum_qty  |  sum_base_price  |   sum_disc_price    |      sum_charge       |       avg_qty       |     avg_price      |        avg_disc
  | count_order
--------------+--------------+-----------+------------------+---------------------+-----------------------+---------------------+--------------------+----------------------
--+-------------
 A            | F            | 377518399 |  566065727797.25 |  537759104278.0656 |  559276670892.116819 | 25.5009751030070973 | 38237.151008958546 | 0.0500065745402432046
3 |    14804077
 N            | F            |   9851614 |   14767438399.17 |   14028805792.2114 |   14590490998.366737 | 25.5224483028409474 | 38257.810660081140 | 0.0499733677376566718
0 |      385998
 N            | O            | 743124873 | 1114302286901.88 | 1058580922144.9638 | 1100937000170.591854 | 25.4980758706893147 | 38233.902923481810 | 0.0500008118211313060
3 |    29144351
 R            | F            | 377732830 |  566431054976.00 |  538110922664.7677 |  559634780885.086257 | 25.5083847896801383 | 38251.219273559761 | 0.0499967923140874204
5 |    14808183
(4 rows)

Time: 9407.291 ms
```

Before parallel acceleration	After parallel acceleration
17456.066 ms	9407.291 ms

22.Query cache

V6.3.7.0 provides the query cache feature to speed up data retrieval by caching query results. This feature improves the performance of database queries in scenarios that require more reads than writes, especially those in which identical queries are frequently repeated.

Precautions

The query cache feature is available only in V6.3.7.0 or later. For information about how to query and update the minor engine version of an AnalyticDB for PostgreSQL instance, see Query the minor engine version and Upgrade the engine version.

The following limits apply when you use the query cache feature in :

- Query cache is supported only when the transaction isolation level is read committed (READ-COMMITTED).
- Query cache is supported only when the Grand Unified Configuration (GUC) parameters rds_uppercase_colname and gp_select_invisible are set to off.
- Cached data can be queried only when all tables in the query have the query cache feature enabled.
- Query cache is supported only when the frontend protocol version of libpq is 3.0 or later.
- If a query within a transaction block has been modified, its query results cannot be stored in the query cache.
- Query cache is not supported for temporary tables, views, materialized views, system tables, unlogged tables, external tables, or volatile or immutable functions.
- Query cache is not supported for queries on child partitioned tables.
- Query cache is not supported for queries when multiple coordinator nodes exist.
- Query cache is not supported if the result set exceeds 7.5 KB in size.
- Query cache is not supported when more than 32 tables are involved in a single query.
- Query cache is not supported if cursors are used in extended queries.

Enable query cache for an instance

By default, query cache is disabled because it is suitable only when a query has high temporal locality. To enable query cache for an instance, 提交工单Submit a ticket to contact the technical support personnel.

Enable or disable query cache for a session

You can use the rds_session_use_query_cache parameter to enable or disable query cache for a session.

Execute the following statement to enable query cache for a session:

```
SET rds_session_use_query_cache = on;
```

Execute the following statement to disable query cache for a session:

```
SET rds_session_use_query_cache = off;
```

Enable or disable query cache for a table

You can use the querycache_enabled parameter to enable or disable query cache for a table.

For a new table, execute the following statement to enable query cache:

```
CREATE TABLE table_name (c1 int, c2 int) WITH (querycache_enabled=on);
```

For a table that did not have query cache enabled when the table was created, execute the following statement to enable query cache:

```
ALTER TABLE table_name SET (querycache_enabled=on);
```

For a table that no longer requires query cache, execute the following statement to disable query cache:

```
ALTER TABLE table_name SET (querycache_enabled=off);
```

Modify the validity period of query cache

When DDL or DML statements are being executed, query results stored in the query cache expire. This prevents the expired query results from being returned. However, uses the multiversion concurrency control (MVCC) mechanism, and the query cache stores only the latest query results. As a result, expired query results are returned in scenarios such as those in which concurrent read and write operations exist.

By default, results stored in the query cache remain valid for 10 minutes. If an identical query is made after the query results have been cached for longer than 10 minutes, the query is performed normally and the cached results for the query are not returned.

To prevent expired query results from being returned, you can Submit a ticket to contact the technical support personnel to modify the validity period of query cache.

Performance evaluation

This section evaluates the performance of query cache in two load scenarios: online transaction processing (OLTP) and online analytical processing (OLAP).

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

OLTP

The following table describes the test results for point queries with indexes.

Scenario	Query cache is not used	Query cache is used
Cache hit rate: 0% Statements used in the point query: 1	1718 TPS	• Without cache replacement: 1,399 TPS • With cache replacement: 915 TPS

Scenario	Query cache is not used	Query cache is used
Cache hit rate: 50% Statements used in the point query: 2	807 TPS	• Without cache replacement: 1,367 TPS • With cache replacement: 877 TPS
Cache hit rate: 100% Statements used in the point query: 1	1718 TPS	11219 TPS

In OLTP scenarios, a normal query has a latency of about 10 ms. In the case that the cache hit rate is 100%, the query performance improves by about 6.5 times when query cache is used. Even if the cache hit rate is 0%, the query performance when query cache is used is not significantly lower than that when query cache is not used. The absolute amount of time required to complete a query with query cache enabled does not change significantly and does not exceed 20 ms when cache replacement exists.

OLAP

The following table describes the test results of queries on 10 GB of data.

Scenario	Query cache is not used	Query cache is used
10 GB TPC-H	1,255 seconds	522 seconds
10 GB TPC-DS	2,813 seconds	1,956 seconds

In OLAP scenarios, the query performance when query cache is used is significantly higher than that when query cache is not used. For example, in the TPC-H test, the performance of Q1 query improves by more than 1,000 times when the cache hit rate is not 0%. In the TPC-DS test, specific query results exceed the maximum size of 7.5 KB allowed for the query cache and are not cached. As a result, the test result does not present a significant performance improvement.

23.Dynamic join filter

The dynamic join filter feature can significantly improve the hash join performance of . This topic describes how to use the dynamic join filter feature.

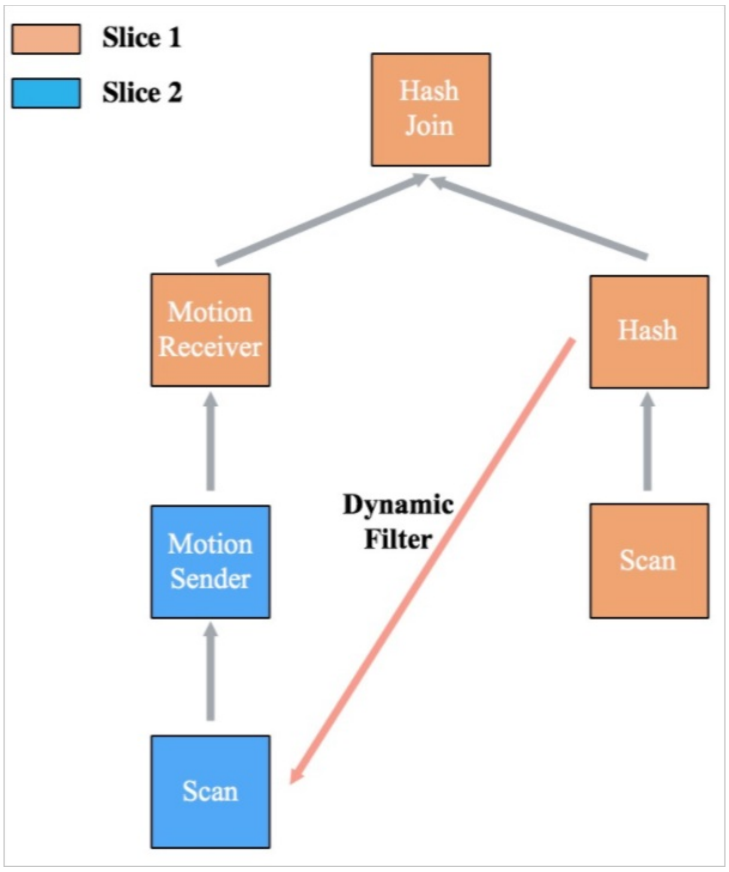
Precautions

- Dynamic join filter is available only in AnalyticDB for PostgreSQL V6.3.8.0 or later. For information about how to update the minor engine version of an AnalyticDB for PostgreSQL instance, see Upgrade the engine version.
- Dynamic join filter is supported for the Legacy query optimizer but not for the ORCA optimizer. For more information about optimizers, see Optimize query performance.
- Dynamic join filter is supported only for the Laser computing engine. For more information about the Laser computing engine, see Use the Laser computing engine.
- Dynamic join filter is not supported for the SQL statements that contain IntiPlan in the execution plan.
- Dynamic join filter is supported for the User Datagram Protocol (UDP) but not for the Transmission Control Protocol (TCP).

Feature description

In , hash joins are commonly used and can implement more than 95% of correlated queries. Data queries may hit a bottleneck when hash joins are used because hash joins involve disk reads and writes, network interaction, and large amounts of computations.

The dynamic join filter feature is supported as of V6.3.8.0. This feature can be used before a hash join to filter out the data that does not match the hash join from the left table by using the dynamically collected join key of the right table. This way, the disk reading, networking, and CPU computing overheads are reduced to improve the hash join performance. The following figure shows the principle of dynamic join filter:



Disable or enable the dynamic join filter feature

By default, the dynamic join filter feature is enabled. You can modify the adbpg_enable_dynamic_join_filter value to disable or enable this feature.

- Execute the following statement to disable the dynamic join filter feature for a session:

```
SET adbpg_enable_dynamic_join_filter TO off;
```

- Execute the following statement to enable the dynamic join filter feature for a session:

```
SET adbpg_enable_dynamic_join_filter TO on;
```

To disable or enable the dynamic join filter feature for an instance, 提交工单Submit a ticket to contact the technical support personnel.

Examples

In this example, the lineitem and part tables in the TPC-H test set are used. The following section uses TPC-H Q17 to compare the query duration before and after the dynamic join filter feature is enabled.

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

1. Generate 1 GB of test data based on the TPC-H benchmark. For more information, see the "Generate test data" section in TPC-H.

2. Execute the following statements to create the lineitem and part tables:

```
CREATE TABLE LINEITEM ( L_ORDERKEY BIGINT NOT NULL, L_PARTKEY INTEGER NOT NULL, L_SUPPK
EY INTEGER NOT NULL, L_LINENUMBER INTEGER NOT NULL, L_QUANTITY NUMERIC(15,2) NOT NULL,
L_EXTENDEDPRICE NUMERIC(15,2) NOT NULL, L_DISCOUNT NUMERIC(15,2) NOT NULL, L_TAX NUMERI
C(15,2) NOT NULL, L_RETURNFLAG CHAR(1) NOT NULL, L_LINESTATUS CHAR(1) NOT NULL, L_SHIPD
ATE DATE NOT NULL, L_COMMITDATE DATE NOT NULL, L_RECEIPTDATE DATE NOT NULL, L_SHIPINSTR
UCT CHAR(25) NOT NULL, L_SHIPMODE CHAR(10) NOT NULL, L_COMMENT VARCHAR(44) NOT NULL ) W
ITH (APPENDONLY=TRUE, ORIENTATION=COLUMN) DISTRIBUTED BY (L_ORDERKEY, L_LINENUMBER); CR
EATE TABLE PART ( P_PARTKEY INTEGER NOT NULL, P_NAME VARCHAR(55) NOT NULL, P_MFGR CHAR(
25) NOT NULL, P_BRAND CHAR(10) NOT NULL, P_TYPE VARCHAR(25) NOT NULL, P_SIZE INTEGER NO
T NULL, P_CONTAINER CHAR(10) NOT NULL, P_RETAILPRICE NUMERIC(15,2) NOT NULL, P_COMMENT
VARCHAR(23) NOT NULL ) WITH (APPENDONLY=TRUE, ORIENTATION=COLUMN) DISTRIBUTED BY (P_PAR
TKEY);
```

3. Execute the following \COPY statements to import 1 GB of test data to the lineitem and part tables:

```
\COPY LINEITEM FROM 'lineitem.tbl' with DELIMITER '|' NULL ''; \COPY PART FROM 'part.tb
l' with DELIMITER '|' NULL '';
```

4. Query the execution duration of a Q17 query when the dynamic join filter feature is disabled.
   i. Disable the dynamic join filter feature.

   ```
   SET adbpg_enable_dynamic_join_filter TO off;
   ```

   ii. Execute a TCP-H Q17 query.

   ```
   SELECT sum(l_extendedprice) / 7.0 as avg_yearly FROM lineitem, part WHERE p_partkey
   = l_partkey and p_brand = 'Brand#54' and p_container = 'SM CAN' and l_quantity < (
   SELECT 0.2 * avg(l_quantity) FROM lineitem WHERE l_partkey = p_partkey );
   ```

   iii. View the execution duration from the query result. The execution duration is 3,468 ms.

   ```
    avg_yearly --------------------- 336452.465714285714 (1 row) Time: 3468.411 ms
   ```

5. Query the execution duration of a Q17 query when the dynamic join filter feature is enabled.
   i. Enable the dynamic join filter feature.

   ```
   SET adbpg_enable_dynamic_join_filter TO on;
   ```

   ii. Execute a TCP-H Q17 query.

   ```
   SELECT sum(l_extendedprice) / 7.0 as avg_yearly FROM lineitem, part WHERE p_partkey
   = l_partkey and p_brand = 'Brand#54' and p_container = 'SM CAN' and l_quantity < (
   SELECT 0.2 * avg(l_quantity) FROM lineitem WHERE l_partkey = p_partkey );
   ```

   iii. View the execution duration from the query result. The execution duration is 305 ms.

   ```
    avg_yearly --------------------- 336452.465714285714 (1 row) Time: 305.632 ms
   ```

After the dynamic join filter feature is enabled, the query performance is significantly improved as the execution duration is reduced from 3,468 ms to 305 ms.