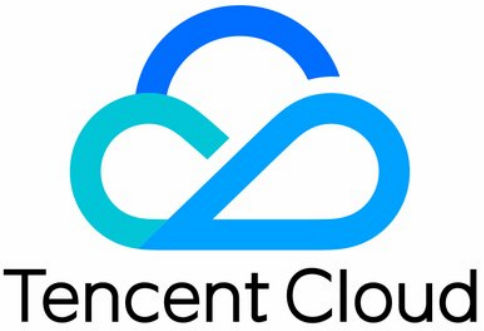
Tencent Cloud TCHouse-P Development Guide



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Development Guide
Data Type

Last updated: 2025-07-02 15:35:13

The database offers a rich set of built-in data types for users. Users can also use the CREATE TYPE command to define new data types. This reference shows all the built-in data types.

Name	Alias	Size	Scope	Description
bigint	int8	8bytes	[-9223372036854775808,9223372036854775807]	Large range of integers
bigserial	serial8	8bytes	[1,9223372036854775807]	Large auto-increment integer
bit [(n)]	-	*n* bits	[bit string constant]	Fixed-length bit string
bit varying [(n)]	varbit	actual number of bits	[bit string constant]	Variable-length bit string
boolean	bool	1byte	true/false, t/f, yes/no, y/n, 1/0	Logical Boolean (true / false)
box	-	32bytes	((x1,y1),(x2,y2))	Rectangle frame in the plane, not allowed in the distribution key column
bytea	-	1 byte + *binary string*	sequence of [octets]	Variable-length binary string
character [(n)]	char [(n)]	1byte + *n*	strings up to *n* characters in length	Fixed-length blank padding
character varying [(n)]	varchar [(n)]	1byte + *string size*	strings up to *n* characters in length	Constrained variable length

cidr	-	12 - 24bytes	-	IPv4 and IPv6 networks
circle	-	24bytes	<(x,y),r> (center and radius)	Circle in the plane, not allowed in the distribution key column
date	-	4bytes	[4713 BC,294,277 AD]	Calendar date (year, month, day)
decimal [(p, s)]	numeric [(p, s)]	variable	no limit	User-specified precision, accurate
double precision	float8 float	8bytes	15 decimal digits precision	Variable precision, inexact
inet	-	12 or 24 bytes	-	IPv4 or IPv6 network address
integer	int, int4	4 bytes	[-2147483648,+2147483647]	Integer type is usually selected
interval [(p)]	-	12 bytes	[-178000000 years,178000000 years]	Time span
json	-	1 byte + json size	json of any length	Unlimited variable length
lseg	-	32 bytes	((x1,y1),(x2,y2))	Segment in the plane, not allowed in the allocation key column
macaddr	-	6 bytes	-	MAC Addresses
money	-	8 bytes	[-92233720368547758.08,+922337203685477 58.07]	Currency amount

path	–	16+16n bytes	[(x1,y1),...]	Geometric path on the plane, not allowed in the distribution key column
point	–	16 bytes	(x,y)	Geometric point on the plane, not allowed in the distribution key column
polygon	–	40+16n bytes	((x1,y1),...)	Closed geometric path in the plane, not allowed in the allocation key column
real	float4	4 bytes	6 decimal digits precision	Variable precision, inaccurate
serial	serial 4	4 bytes	[1,2147483647]	Auto-increment integer
smallint	int2	2 bytes	[-32768,+32767]	Small range of integers
text[1]	–	1 byte + *string size*	strings of any length	Unlimited variable length
time [(p)] [without time zone]	–	8 bytes	[00:00:00[.000000],24:00:00[.000000]]	Time of only one day
time [(p)] with time zone	timetz	12bytes	[00:00:00+1359,24:00:00-1359]	Time of only one day, with timezone
timestamp [(p)] [without time zone]	–	8 bytes	[4713 BC,294,277 AD]	Date and time
timestamp [(p)] with time zone	timestamptz	8 bytes	[4713 BC,294,277 AD]	Datetime, with timezone

uuid	–	32 bytes	–	Universal Unique Identifier according to RFC 4122, ISO/IEC 9834-8:2005
xml[1]	–	1 byte + *xml size*	xml of any length	Unlimited variable length
txid_snapshot	–	–	–	User-level transaction ID snapshot

Functions and Operators
Logical Operators

Last updated: 2024-08-22 16:19:14

Common logical operators are AND, OR, NOT, with three result values TRUE, FALSE, and NULL, where NULL represents an unknown value.

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

Comparison Operators

Last updated: 2024-08-22 16:19:44

Common comparison operators and functions are as listed below:

Operator	Feature
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<> or !=	Not equal to

Mathematical Functions and Operators

Last updated: 2024-08-22 16:17:28

The database provides many mathematical operators, as shown in the table below. Bitwise operators can only be applied to integer types, while other operators can be applied to all numeric types.

Operator	Description	Sample code	Result
+	Addition	2 + 3	5
-	Subtraction	2 - 3	-1
*	Multiplication	2 * 3	6
/	Division	4 / 2	2
%	Modulo	5 % 4	1
^	Exponentiation	2.0 ^ 3.0	8
|/	Square root	|/ 25.0	5
||/	Cube root	||/ 27.0	3
!	Factorial	5 !	120
!!	factorial (prefix operator)	!! 5	120
@	Absolute value	@ -5.0	5
&	Bitwise AND	91 & 15	11
|	Bitwise OR	32 | 3	35
#	Bitwise XOR	17 # 5	20
~	Bitwise NOT	~1	-2
<<	Bitwise Shift Left	1 << 4	16
>>	Bitwise Shift Right	8 >> 2	2

The database also provides mathematical functions, as shown in the table below.

Function	Return Type	Description	Sample code	Result
abs(x)	Same as parameter x type	Absolute value	abs(-17.4)	17.4
cbrt(x)	Same as parameter x type	Cube root	cbrt(27.0)	3
ceil(x)	Same as parameter x type	Smallest integer not less than x	ceil(-42.8)	-42
ceiling(x)	Same as parameter x type	Alias function for ceil	ceiling(-95.3)	-95
degrees(x)	Same as parameter x type	Convert radians to degrees	degrees(0.5)	28.6478897565412
exp(x)	Same as parameter x type	Exponential function with base e	exp(1.0)	2.71828182845905
floor(x)	Same as parameter x type	Largest integer not greater than x	floor(-42.8)	-43
ln(x)	Same as parameter x type	Logarithm with base e	ln(2.0)	0.693147180559945
log(x)	Same as parameter x type	Logarithm base 10	log(100.0)	2
log(b numeric, x numeric)	numeric	Logarithm of x with base b	log(2.0, 64.0)	6.00000000
mod(y, x)	(same as argument types)	y%x	mod(9,4)	1

pi()	dp	"π" Constant	pi()	3.14159265358979
power(a dp, b dp)	dp	a raised to the power of b	power(9.0, 3.0)	729
radians(dp)	Input type consistent	Degrees to radians	radians(45.0)	0.785398163397448
random()	dp	Random value in range 0.0 <= x < 1.0	random()	–
round(dp or numeric)	Input type consistent	Round off	round(42.4)	42
round(v numeric, s int)	numeric	Retain s decimal places, round off	round(42.4382, 2)	42.44
setseed(dp)	void	Set seed for random function random()	setseed(0.54823)	–
sign(dp or numeric)	(-1, 0, +1)	Get sign bit	sign(-8.4)	-1
sqrt(dp or numeric)	Input type consistent	Square root	sqrt(2.0)	1.4142135623731
trunc(dp or numeric)	Input type consistent	Clear decimal places to zero	trunc(42.8)	42
trunc(v numeric, s int)	numeric	Keep decimal places to s; excess digits will be cleared to zero	trunc(42.4382, 2)	42.43

The database also supports trigonometric functions, as shown in the table below.

Function	Description
acos(x)	inverse cosine
asin(x)	inverse sine
atan(x)	inverse tangent
atan2(y, x)	inverse tangent of y/x

cos(x)	cosine
cot(x)	cotangent
sin(x)	sine
tan(x)	tangent

String Functions and Operators

Last updated: 2024-08-22 16:20:30

This section mainly discusses some string operation functions. See the table below for details.

Function	Returned values	Description	Sample code	Result
string || string	text	String concatenation	'Post' || 'greSQL'	PostgreSQL
string || non-string or non-string || string	text	Concatenation between strings and non-strings	'Value: ' || 42	Value: 42
bit_length(string)	int	Number of bits in a string	bit_length('jose')	32
char_length(string) or character_length(string)	int	Character count of a string	char_length('jose')	4
lower(string)	text	Convert to lowercase characters	lower('TOM')	tom
octet_length(string)	int	Byte count of a string	octet_length('jose')	4
overlay(string placing string from int [for int])	text	Replace substring	overlay('Txxxxas' placing 'hom' from 2 for 4)	Thomas

position(substring in string)	int	Position of substring in string	position('om' in 'Thomas')	3
substring(string [from int] [for int])	text	Extract substring	substring('Thomas' from 2 for 3)	hom
substring(string from pattern)	text	Regular expression matching of substring	substring('Thomas' from '...$')	mas
trim([leading | trailing | both] [characters] fromstring)	text	Remove substring	trim(both 'x' from 'xTomxx')	Tom
upper(string)	text	Convert to uppercase letters	upper('tom')	TOM

The database also provides some other string functions to support string manipulation. See the table below.

Function	Returned values	Feature	Sample code	Result
ascii(string)	int	Get the ASCII code of the first character	ascii('x')	120
btrim(string text [, characters text])	text	Remove the longest sequence from the beginning and end (but not the middle) of the string that only contains characters from	btrim('xyxtrimyyx', 'xy')	trim

		characters (default is whitespace)		
chr(int)	text	Convert ASCII code to character	chr(65)	A
convert(string bytea, src_encodingname, dest_encodingname)	bytea	Change encoding using the specified conversion name	convert('text_in_utf8', 'UTF8', 'LATIN1')	–
convert_from(string bytea,src_encoding name)	text	Convert to the database's default encoding format	convert_from('text_in_utf8', 'UTF8')	text_in_utf8represented in the current database encoding
convert_to(string text, dest_encodingname)	bytea	Convert data to a specified encoding format	convert_to('some text', 'UTF8')	–
decode(string text, type text)	bytea	Decoding function	decode('MTIzAAE=', 'base64')	123\000\001
encode(data bytea, type text)	text	Encoding function	encode(E'123\000\001', 'base64')	MTIzAAE=
initcap(string)	text	Capitalize the first letter of each word, with the rest in lowercase	initcap('hi THOMAS')	Hi Thomas
length(string)	int	String length	length('jose')	4
length(stringbytea, encoding name)	int	Length of the string in the specified encoding format	length('jose', 'UTF8')	4
ltrim(string text [, characters text])	text	Remove the longest sequence from the beginning (but not the	ltrim('zzzytrim', 'xyz')	trim

		middle) of the string that only contains characters from characters (default is whitespace)		
md5(string)	text	Compute the MD5 value of the string	md5('abc')	900150983cd24fb0d6963f7d28e17f72
pg_client_encoding()	name	Get the client's encoding format	pg_client_encoding()	SQL_ASCII
regexp_replace(string text, patterntext, replacement text [, flags text])	text	–	regexp_replace('Thomas', '.[mN]a.', 'M')	ThM
repeat(string text, n int)	text	Repeat the string n times	repeat('Pg', 4)	PgPgPgPg
replace(string text, from text, totext)	text	Replace string	replace('abcdefabcdef', 'cd', 'XX')	abXXefabXXef
rtrim(string text [, characters text])	text	Remove the longest sequence from the end (but not the middle) of the string that only contains characters from characters (default is whitespace)	rtrim('trimxxxx', 'x')	trim
strpos(string, substring)	int	Get the position of the substring	strpos('high', 'ig')	2
substr(string, from [, count])	text	Get substring	substr('alphabet', 3, 2)	ph
to_ascii(string text [,	text	Convert to ASCII code format	to_ascii('Karel')	Karel

encodingtext])				
to_hex(number int or bigint)	te xt	Convert number to hexadecimal	to_hex(21 47483647)	7fffffff

Pattern Matching

Last updated: 2024-08-22 16:21:23

LIKE

The database supports various pattern matching methods. The first one is LIKE or NOT LIKE.

Sample code	Result
‘abc’ LIKE ‘abc’	true
‘abc’ LIKE ‘abcz’	false
‘abc’ LIKE ‘a%’	true
‘abc’ LIKE ‘_a_’	false
‘abc’ LIKE ‘a_’	false
‘abc’ LIKE ‘a’	false

SIMILAR TO

The second pattern matching method is SIMILAR TO, which is supported by the database. It works similar to LIKE, but supports more matching syntax. See the table below for details.

Syntax	Description	Sample code	Result
|	Supports optional matching	'abc' SIMILAR TO '%(b|d|c|e)%'	True
*	The preceding element repeated 0 or more times	'abc' SIMILAR TO 'abcd*'	True
'abc' SIMILAR TO 'abc*'	true	-	-

'abc' SIMILAR TO 'ab*'	false	-	-
+	The preceding element repeated 1 or more times	'abc' SIMILAR TO 'abc+'	True
'abc' SIMILAR TO 'abcc+'	false	-	-

POSIX Regular Expressions

The database also supports POSIX regular expressions, which provide more powerful functions than LIKE and SIMILAR TO operators. POSIX regular expressions supported functions are shown in the table below.

Function	Feature	Sample code	Result
substring(string from pattern)	Extract a substring from string according to the regular pattern	substring('foobar' from 'o.b')	oob
regexp_replace(source, pattern, replacement [, flags])	Replace the matched string in the source	regexp_replace('foobarbaz', 'b..', 'X')	fooXX
regexp_matches(string, pattern [, flags])	The function returns a text array, which consists of all captured substrings obtained by matching a POSIX regular expression pattern	regexp_matches('foobarbequebaz', '(bar)(beque)');	{bar,beque}
regexp_split_to_table(string, pattern [, flags])	The function splits a string using a POSIX regular expression pattern as a delimiter	SELECT regexp_split_to_table('the quick brown fox jumped', E'\\s+') ;	the quick brown fox jumped

regexp_split_to_array(string, pattern [, flags])	Similar to regexp_split_to_table, it is a regular expression split function, but it returns the result as a text array	SELECT regexp_split_to_array('the quick brown fox jumped', E'\\s+');	{the,quick,brown,fox,jumped}

Datetime Functions and Operators

Last updated: 2024-08-22 16:22:00

The table below shows the common date and time functions and operators supported by the database.

Operator	Sample code	Result
+	date '2001-09-28' + integer '7'	date '2001-10-05'
+	date '2001-09-28' + interval '1 hour'	timestamp '2001-09-28 01:00:00'
+	date '2001-09-28' + time '03:00'	timestamp '2001-09-28 03:00:00'
+	interval '1 day' + interval '1 hour'	interval '1 day 01:00:00'
+	timestamp '2001-09-28 01:00' + interval '23 hours'	timestamp '2001-09-29 00:00:00'
+	time '01:00' + interval '3 hours'	time '04:00:00'
-	- interval '23 hours'	interval '-23:00:00'
-	date '2001-10-01' - date '2001-09-28'	integer '3'
-	date '2001-10-01' - integer '7'	date '2001-09-24'
-	date '2001-09-28' - interval '1 hour'	timestamp '2001-09-27 23:00:00'
-	time '05:00' - time '03:00'	interval '02:00:00'
-	time '05:00' - interval '2 hours'	time '03:00:00'
-	timestamp '2001-09-28 23:00' - interval '23 hours'	timestamp '2001-09-28 00:00:00'
-	interval '1 day' - interval '1 hour'	interval '1 day -01:00:00'
-	timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00'	interval '1 day 15:00:00'
*	900 * interval '1 second'	interval '00:15:00'

*	21 * interval '1 day'	interval '21 days'
*	double precision '3.5' * interval '1 hour'	interval '03:30:00'
/	interval '1 hour' / double precision '1.5'	interval '00:40:00'

The table below lists common time/date functions.

Function	Returned values	Description	Sample code	Result
age(timestamp, timestamp)	interval	Calculates time difference	age(timestamp '2001-04-10', timestamp '1957-06-13')	43 years 9 mons 27 days
age(timestamp)	interval	Calculates time difference	age(timestamp '1957-06-13')	43 years 8 mons 3 days
current_date	date	Current date	select current_date;	2019-02-18
current_time	time with time zone	Current time of day	select current_time;	16:42:59.991189 +08
current_timestamp	timestamp with time zone	Current timestamp	select current_timestamp;	2019-02-18 16:43:20.167284 +08
date_part(text, timestamp)	double precision	Gets a part of datetime	date_part('hour', timestamp '2001-02-16 20:38:40')	20
date_part(text, interval)	double precision	Gets a part of datetime	date_part('month', interval '2 years 3 months')	3
date_trunc(text, timestamp)	timestamp	Clears the specified part of datetime	date_trunc('hour', timestamp '2001-02-16 20:38:40')	2001-02-16 20:00:00
extract(field from	double precision	Similar to date_part	extract(hour from timestamp '2001-02-16 20:38:40')	20

timestamp)				
extract(field from interval)	double precision	Similar to date_part	extract(month from interval '2 years 3 months')	3
localtime	time	Gets local time	select localtime;	16:56:09.339026
localtimestamp	timestamp	Local date and time	select localtimestamp;	2019-02-18 16:56:42.331012
now()	timestamp with time zone	Gets current time	select now();	2019-02-18 16:56:58.843212 +08
timeofday( )	text	Current date and time	select timeofday();	Mon Feb 18 16:57:27.677262 2019 CST

Geometric Functions and Operators

Last updated: 2024-08-22 16:22:31

The geometric types supported by the database include point, box, lseg, line, path, polygon, circle. Therefore, the database also provides a series of geometric functions. See the table below for details.

Geometric Operators

Operator	Description	Sample code	Result
+	Translate	box '((0,0),(1,1))' + point '(2.0,0)'	(3,1), (2,0)
-	Translate	box '((0,0),(1,1))' - point '(2.0,0)'	(-1,1), (-2,0)
*	Stretch/Rotate	box '((0,0),(1,1))' * point '(2.0,0)'	(2,2), (0,0)
/	Stretch/Rotate	box '((0,0),(2,2))' / point '(2.0,0)'	(1,1), (0,0)
#	Intersection of two geometries	box '((1,-1),(-1,1))' # box '((1,1), (-1,-1))'	(1,1), (-1,-1)
@-@	Length or Perimeter of the geometry	@-@ path '((0,0),(1,0))'	2
@@	Center	@@ circle '((0,0),10)'	(0,0)
##	The closest point of the first geometry to the second geometry	point '(0,0)' ## lseg '((2,0),(0,2))'	(1,1)
<->	Distance between geometries	circle '((0,0),2)' <-> circle '((4,0),1)'	1
&&	Whether two geometries intersect	box '((0,0),(1,2))' && box '((0,0), (2,3))'	t
<<	Whether geometry 1 is strictly to the left of	circle '((0,0),1)' << circle '((5,0),1)'	t

	geometry 2		
>>	Whether geometry 1 is strictly to the right of geometry 2	circle '((5,0),1)' >> circle '((0,0),1)'	t
&<	Whether the right edge of geometry 1 does not exceed the right edge of geometry 2	box '((0,0),(1,1))' &< box '((0,0),(2,2))'	t
&>	Does the rightmost edge of Graphic 1 not exceed the leftmost edge of Graphic 2	box '((0,0),(3,3))' &> box '((0,0),(2,2))'	t
<<|	Is Graphic 1 strictly below Graphic 2	box '((0,0),(2,2))' <<| box '((4,5),(6,6))'	t
|>>	Is Graphic 1 strictly above Graphic 2	box '((5,6),(7,7))' |>> box '((0,0),(4,4))'	t
<^	Is Graphic 1 below Graphic 2 (contact allowed)	box '((5,6),(7,7))' <^ box '((0,0),(4,4))'	f
>^	Is Graphic 1 above Graphic 2 (contact allowed)	box '((5,6),(7,7))' >^ box '((0,0),(4,4))'	t
?#	Intersect	lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))'	t
?-	Is it horizontal	?- lseg '((-1,0),(1,0))'	t
?-	Are the two graphics horizontally aligned	point '(1,0)' ?- point '(0,0)'	t
?|	Is it vertical	?| lseg '((-1,0),(1,0))'	f
?|	Are the two graphics vertically aligned	point '(1,2)' ?| point '(1,1)'	t
?-|	Are the two lines vertical	lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))'	t
?||	Are the two lines parallel	lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))'	t
@>	Does Graphic 1 include Graphic 2	circle '((0,0),2)' @> circle '((0,0),1)"	t

<@	Is Graphic 1 included by Graphic 2	point '(1,1)' <@ circle '((0,0),2)'	t
~=	Are they identical	polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))'	t

Geometric Functions

Function	Return Type	Description	Sample code	Result
area(object)	double precision	Area	area(box '((0,0),(1,2))')	2
center(object)	point	Centroid	center(box '((0,0),(1,2))')	(0.5, 1)
diameter(circle)	double precision	Circle Diameter	diameter(circle '((0,0),2.0)')	4
height(box)	double precision	High	height(box '((0,0),(2,3))')	3
isclosed(path)	boolean	Is Shape Closed	isclosed(path '((0,0),(1,1),(2,0))')	t
isopen(path)	boolean	Is Shape Open	isopen(path '[(0,0),(1,1),(2,0)]')	t
length(object)	double precision	Shape Length	length(path '((-1,0),(1,0))')	2
npoints(path)	int	Number of Vertices	npoints(path '[(0,0),(1,1),(2,0)]')	3
npoints(polygon)	int	Number of Vertices	npoints(polygon '((1,1),(0,0))')	2
radius(circle)	double precision	Circle Radius	radius(circle '((0,0),2.0)')	2
width(box)	double precision	Horizontal Length	width(box '((0,0),(3,2))')	3

Geometric Type Conversion Functions

Function	Return Type	Description	Sample code	Returned values
box(circle)	box	Circle to Rectangle	box(circle '((0,0),3.0)')	(2.12132034355964,2.12132034355964), (-2.12132034355964,-2.12132034355964) (1 row)
box(point, point)	box	Point to Rectangle	box(point '(0,0)', point '(2,2)')	(2,2),(0,0)
box(polygon)	box	Polygon to Rectangle	box(polygon '((0,0),(1,1),(2,0))')	(2,1),(0,0)
circle(box)	circle	Convert Rectangle to Circle	circle(box '((0,0),(2,2))')	<(1,1),1.4142135623731>
circle(point, double precision)	circle	Convert Center and Radius to Circle	circle(point '(0,0)', 3.0)	<(0,0),3>
circle(polygon)	circle	Convert Polygon to Circle	circle(polygon '((0,0),(1,1),(2,0))')	<(1,0.333333333333333),0.924950591148529>
lseg(box)	lseg	Convert Rectangle to	lseg(box'((-1,0),(1,0))')	[(1,0),(-1,0)]

		Segment		
lseg(point, point)	lseg	Convert Multiple Points to Segment	lseg(point '(-1,0)', point '(1,0)')	[(-1,0),(1,0)]
path(polygon)	path	polygon to path	path(polygon '((0,0),(1,1),(2,0))')	((0,0),(1,1),(2,0))
point(double precision, double precision)	point	construct point	point(23.4, -44.5)	(23.4,-44.5)
point(box)	point	Center of Graphics	point(box '((-1,0),(1,0))')	(0,0)
point(circle)	point	Center of Circle	point(circle '((0,0),2.0)')	(0,0)
point(lseg)	point	Center of Segment	point(lseg '((-2,0),(2,0))')	(0,0)
point(polygon)	point	Center of Polygon	point(polygon '((0,0),(1,1),(2,0))')	(1,0.333333333333333)
polygon(box)	polygon	Convert Rectangle to a Four-	polygon(box '((0,0),(1,1))')	((0,0),(0,1),(1,1),(1,0))

		Point Polygon		
polygon(circle)	polygon	Convert Circle to a 12-Point Polygon	polygon(circle '((0,0),2.0)')	-
polygon(npts, circle)	polygon	Convert Circle to a Polygon with n Points	polygon(12, circle '((0,0),2.0)')	((-2,0), (-0.618033988749895,1.902113303259031), (1.61803398874989,1.175570504584 95), (1.6180339887499,-1.175570504584 95), (-0.618033988749894,-1.902113303259031))
polygon(path)	polygon	Convert Path to Polygon	polygon(path '((0,0),(1,1),(2,0))')	((0,0),(1,1),(2,0))

Sequence Manipulation Functions

Last updated: 2024-08-22 16:23:32

The database supports sequences. Sequence objects are special single-row tables, created using CREATE SEQUENCE. A sequence is typically used to generate unique table data. The functions of a sequence are as follows. For example, to create a sequence: `CREATE SEQUENCE myseq START 101;`.

Function	Returned values	Description	Result
nextval('myseq')	bigint	The sequence auto-increments and returns the latest value	101
setval('myseq',102)	bigint	Sets sequence's current value	102

Condition expression

Last updated: 2024-08-22 16:23:50

CASE

The database supports CASE expressions, similar to the IF/ELSE feature in other languages. Example:

```
SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
    FROM test;
```

COALESCE

COALESCE returns the first non-NULL value among its parameters. If all parameters are NULL, it returns NULL.

```
SELECT COALESCE(null,1,2,null) ;

 coalesce

----------

        1
```

NULLIF

NULLIF(value1, value2) returns: if value1 and value2 are equal, it returns NULL. Otherwise, it returns value1.

GREATEST and LEAST

These two functions return the largest or smallest value from a list of values, respectively. When all parameters are NULL, it returns NULL.

Aggregate function

Last updated: 2024-08-22 16:24:26

The aggregation functions of the database compute a return value from a series of values. The table below lists some built-in aggregation functions.

Function	Parameter Type	Return Type	Description
avg(expression)	All numeric types	Numeric type	Average value
bit_and(expression)	smallint, int, bigint, bit	Same as parameter type	Bitwise AND of all non-null values
bit_or(expression)	smallint, int, bigint, bit	Same as parameter type	Bitwise OR of all non-null values
bool_and(expression)	bool	bool	Returns true if all values are true, otherwise returns false
bool_or(expression)	bool	bool	Returns true if at least one value is true, otherwise returns false
count(*)	-	bigint	Number of returned rows
count(expression)	any	bigint	Returns the sum of all values where the expression is non-null
every(expression)	bool	bool	Same as bool_and
max(expression)	array,numeric,string, date/time type	Same as param	Maximum value in input parameters

		eter type	
min(expression)	array, numeric, string, date/time type	Same as parameter type	Minimum value in input parameters
sum(expression)	smallint, int, bigint, real, double precision,numeric, interval	-	Sum of all input parameters

Subquery Expressions

Last updated: 2024-08-22 16:24:55

The database also supports subquery expressions. The subquery return values listed below are TRUE/FALSE.
The data in the original table t1 is as follows:

```
select * from t1;
 a1
----
  1
  3
  2
```

EXISTS/NOT EXISTS

The parameter of EXISTS is a SELECT statement, or a subquery. The system performs operations on the subquery to determine if it returns any rows. If it returns at least one row, the result of EXISTS is TRUE; if the subquery returns no rows, the result of EXISTS is FALSE. This subquery will generally only run long enough to determine whether at least one row is returned, not all the way to completion. Therefore, the content of the results returned by the subquery is usually not what we are concerned with, so the optimized writing style is: `...WHERE EXISTS(SELECT 1 FROM ...)` .

Sample code	Result
select a1 from t1 where EXISTS(select a1 from t1 where a1>2);	a1 ---- 2 1 3
select a1 from t1 where EXISTS(select a1 from t1 where a1>3);	a1 ----
select a1 from t1 where EXISTS(select a1 from t1 where a1>1) and a1>2;	a1 ---- 3

IN/NOT IN

expression IN (subquery), where the right side is a subquery enclosed in parentheses, and it must return only one field. The left expression performs one calculation and comparison per row of the subquery results. If any equal row is found in the subquery, the result of IN is TRUE. If no equal rows are found, the result is FALSE (including when the subquery returns no rows).

NULL values in expressions or subquery rows follow SQL rules when dealing with Boolean values and NULL combinations. If corresponding fields in both rows are equal and non-null, the rows are equal; if any corresponding fields are unequal but non-null, the rows are unequal; otherwise, the result is unknown (NULL). If each result row is either unequal or NULL, and there is at least one NULL, then the result of IN is NULL.
Pay special attention to subqueries in NOT IN, if there are NULL values, it directly returns FALSE.

Sample code	Result
select * from t1 where a1 in (select a1 from t1 where a1 > 2);	a1 ---- 3 (1 row)
select * from t1 where a1 in (4,5,6);	a1 ---- (0 rows)
select * from t1 where a1 in (2,5,6);	a1 ---- 2 (1 row)

ANY/SOME

expression operator SOME/ANY (subquery), where the right side is a subquery and must return only one field value. The left expression uses the operator to perform one calculation and comparison per row of the subquery result, and the result must be a Boolean value. If at least one true value is obtained, the result of ANY is TRUE. If all false values are obtained, the result is FALSE (including when the subquery returns no rows). SOME is a synonym of ANY. IN and ANY can be used as equivalent substitutions.
Similar to EXISTS, this comparison with the subquery results will generally only run long enough to determine if it is TRUE, not all the way to completion.

Sample code	Result
`select * from t1 where a1 < any(select a1 from t1 where a1 <3);`	a1 ---- 1 (1 row)
`select * from t1 where a1 < some(select a1 from t1 where a1 <4);`	a1 ---- 1 2 (2 rows)
`select * from t1 where a1 < any(select a1 from t1 where a1 <1);`	a1 ---- (0 rows)

ALL

expression operator ALL (subquery), where the right side is a subquery enclosed in parentheses and must return only one field. The left expression uses the operator to perform

one calculation and comparison per row of the subquery result, and the result must be a Boolean value. If all true values are obtained, the result of ALL is TRUE (including when the subquery returns no rows). If at least one false value is obtained, the result is FALSE.
NOT IN is equivalent to ALL.
Similar to EXISTS, this comparison with the subquery results will generally only run long enough to determine if it is FALSE, not all the way to completion.
If the subquery returns NULL, it directly returns TRUE.

Sample code	Result
`select * from t1 where a1 < all(select a1 from t1 where a1 <1);`	a1 ---- 1 3 2 (3 rows)
`select * from t1 where a1 < all(select a1 from t1 where a1 >2);`	a1 ---- 2 1 (2 rows)

Comparison of Row Value and Array

Last updated: 2024-08-22 16:25:25

Operator	Description	Sample code	Result
IN	Consistent with Subquery IN	select * from t1 where a1 in (2,5,6);	a1 ---- 2 (1 row)
NOT IN	Consistent with Subquery NOT IN	select * from t1 where a1 not in (2,5,6);	a1 ---- 1 3 (2 rows)
ANY/SOME	If at least one comparison result is TRUE, it returns TRUE; otherwise, it returns FALSE	select * from t1 where a1 < any(array[1,3]);	a1 ---- 2 1 (2 rows)
ALL	If all comparison results are TRUE, it returns TRUE; otherwise, it returns FALSE	select * from t1 where a1 < ALL(array[3]);	a1 ---- 2 1 (2 rows)

Sequence Functions

Last updated: 2024-08-22 16:25:43

The sequence functions provided by the database return multiple values as listed below.

Function	Parameter Type	Return Type	Description	Sample code
generate_series(start, stop)	Integer	Return a series of values	A series of values starting from start, auto-incrementing to stop	select * from generate_series(2,4); generate_series ---------- 2 3 4 (3 rows)
generate_series(start, stop, step)	Integer	Return a series of values	A series of values starting from start, auto-incrementing to stop with a step length of step	select * from generate_series(5,1,-2); generate_series ---------- 5 3 1 (3 rows)

System Information Functions

Last updated: 2024-08-22 16:26:07

The system information functions provided by the database can retrieve session and system information. See the table below for details.

Function name	Returned values	Description
current_database()	name	Name of current database
current_schema()	name	Current schema name
current_schemas(boolean)	name[]	All schemas set by search_path
current_user	name	User
inet_client_addr()	inet	Current client address (valid in remote connection mode)
inet_client_port()	int	Current client port (valid in remote connection mode)
inet_server_addr()	inet	IP address of the server that accepts connections (valid in remote connection mode)
inet_server_port()	int	Port of the server that accepts connections (valid in remote connection mode)
pg_postmaster_start_time()	timestamp with time zone	Server start time
session_user	name	Session username
user	name	Equivalent to current_user
version()	text	PostgreSQL version

System Management Functions

Last updated: 2024-08-22 16:26:29

Configure Settings Function

Below are functions to query and modify runtime configuration parameters, see the table below.

Function	Returned values	Description	Sample code	Result
current_setting(setting_name)	text	Current Parameter Value	SELECT current_setting('datestyle');	current_setting ---------- ISO, MDY
set_config(setting_name, new_value, is_local)	text	Set parameter values and return the latest values	SELECT set_config('log_statement_stats', 'off', false);	set_config ---------- off

Service Signal Function

Function	Returned values	Description

pg_cancel_backend(pid int)	boolean	Cancel a current backend query
pg_reload_conf()	boolean	Trigger a configuration file reload
pg_rotate_logfile()	boolean	Scroll server log files. Useful only for built-in log collectors; others are not useful as they do not manage log subprocesses

Database Object Size Function

Function	Returned values	Description	Sample code	Result
pg_column_size(any)	int	Get the storage space in bytes	select pg_column_size('ddewewe');	8
pg_database_size(oid)	bigint	Specify the database size	select oid,* from pg_database; select pg_database_size(16384);	pg_database_size ------------------ 127632410
pg_database_size(name)	bigint	Specify the database size	select pg_database_size('gpperfmon');	127632410
pg_relation_size(oid)	bigint	Get the size of specified table or index	select pg_relation_size(17787);	pg_relation_size ------------------ 65536
pg_relation_size(text)	bigint	Get the size of tables or indexes	select pg_relation_size('t1');	pg_relation_size ------------------ 65536

pg_size_pretty(bigint)	text	Convert the number of bytes to a formatted size	select pg_size_pretty(122212121);	pg_size_pretty ---------- 117 MB
pg_tablespace_size(oid)	bigint	Get the size of specified tablespace	select oid,* from pg_tablespace ; select pg_tablespace_size(1663);	pg_tablespace_size ---------- 262275170
pg_tablespace_size(name)	bigint	Get the size of specified tablespace	select pg_tablespace_size('pg_default');	pg_tablespace_size ---------- 262275170
pg_total_relation_size(oid)	bigint	Disk space occupied by the specified table, including indexes and data	select oid,relname from pg_class where relname='t1'; select pg_total_relation_size(17787);	pg_total_relation_size ---------- 65536 (1 row)
pg_total_relation_size(text)	bigint	Disk space occupied by the specified table, including indexes and data	select pg_total_relation_size('t1');	pg_total_relation_size ---------- 65536

DDL Syntax List

Last updated: 2024-08-22 16:27:08

Defining Database

A database is a repository for organizing, storing, and managing data. Database Definition mainly includes creating databases, modifying database attributes, and deleting databases. For the related SQL statements, please refer to the table below.

Database Definition SQL

Feature	Related SQL
Create a database	CREATE DATABASE
Modifying database attributes	ALTER DATABASE
Dropping a Database	DROP DATABASE

Schema Definition

A schema is a collection of database objects primarily used to control access to database objects. For the related SQL statements, please refer to the table below.

Schema Definition SQL

Feature	Related SQL
Creating Schema	CREATE SCHEMA
Modifying schema attributes	ALTER SCHEMA
Deletion Mode	DROP SCHEMA

Table Definition

A table is a special data structure in a database used to store data objects and the relationships between objects. For the related SQL statements, please refer to the table below.

Table Definition SQL

Feature	Related SQL
Create a table	CREATE TABLE
Modifying table attributes	ALTER TABLE
Deleting a table	DROP TABLE

> **Note**
> A single ALTER statement supports only one DDL operation. Executing multiple DDL operations simultaneously may corrupt index metadata.

Definition of Partition Table

A partition table is a logical table where data is stored by regular tables and is primarily used to improve query performance. For the related SQL statements, please refer to the table below.

Partition Table Definition Related SQL

Feature	Related SQL
Creating a Partition Table	CREATE TABLE ···.PARTITION BY···
Creating a Partition	ALTER TABLE ···ADD PARTITION···
Modifying Partition Table Properties	ALTER TABLE ···RENAME/SPLIT PARTITION···
Deleting a Partition	ALTER TABLE ···.DROP PARTITION ···
Deleting a table	DROP TABLE

> **Note**
> A single ALTER statement supports only one DDL operation. Executing multiple DDL operations simultaneously may corrupt index metadata.

Definition of Index

An index is a structure that orders values in one or more columns of a database table. Using an index allows for fast access to specific information within a database table. For the related

SQL statements, please refer to the table below.

Index Definition Related SQL

Feature	Related SQL
Creating an Index	CREATE INDEX
Modifying Index Properties	ALTER INDEX…
Deleting an Index	DROP INDEX
Reindexing	REINDEX

> **Note**
> A single ALTER statement supports only one DDL operation. Executing multiple DDL operations simultaneously may corrupt index metadata.

Role Definition

A role is used to manage permissions. From a database security perspective, all management and operational permissions can be divided into different roles. For related SQL statements, please refer to the table below.

Role Definition Related SQL

Feature	Related SQL
Creates role	CREATE ROLE
Modifying Role Attributes	ALTER ROLE…
Deletes role	DROP ROLE

User Definition

A user is a database role with default LOGIN permissions. For related SQL statements, please refer to the table below.

User Related SQL

Feature	Related SQL

Creating user	CREATE USER
Modifying User Attributes	ALTER USER
Deleting user	DROP USER

Creating function

Feature	Related SQL
Creating function	CREATE FUNCTION
Modifying Function Attributes	ALTER FUNCTION
Deleting function	DROP FUNCTION

View Definition

A view is a virtual table derived from one or more base tables. It can be used to control user data access. Please refer to the table below.

Feature	Related SQL
Creating View	CREATE VIEW
Dropping View	DROP VIEW

Operating Session

The connection established between a user and the database is called a session. Please refer to the table below.

Feature	Related SQL
Modifying Session	ALTER SESSION

Definition Resource Queue

The Load Group is a system table used by the load management module. It is mainly used to specify the number of concurrent jobs that can run within the associated resource pool. For the SQL statements involved, please refer to the table below.

Feature	Related SQL

Creating Resource Queue	CREATE RESOURCE QUEUE
Delete Resource Queue	DROP RESOURCE QUEUE
Modifying Resource Queue	ALTER RESOURCE QUEUE

DML Syntax List

Last updated: 2024-08-22 16:30:41

DML (Data Manipulation Language) is used to manipulate data in database tables, such as Insert, Update, Inquiry, and Delete.

Inserting Data

Insert one or more rows into a table.

Feature	Related SQL
Inserting Data	INSERT

Modifying Data

Modify one or more rows in a table.

Feature	Related SQL
Modifying Data	UPDATE

Querying Data

Query data in a table that meets specified conditions.

Feature	Related SQL
Querying Data	SELECT

Deleting Data

You can delete one or more specified rows, or clear an entire table.

Feature	Related SQL
Deleting Data	DELECT
Clearing data	TRUNCATE

Copying data

Copy data between a file and a table.

Feature	Related SQL
Copying data	COPY

DCL Syntax List

Last updated: 2024-08-22 16:31:05

DCL(Data Control Language), used to set or change database user or role permissions.

Authorization

It defines access privileges.

Feature	Related SQL
Authorization	GRANT

Revoke Permissions

Revoke specified permissions.

Feature	Related SQL
Reclaim Authorization	REVOKE

Basic Database Development Data Type

Last updated: 2024-08-22 16:56:42

Numeric Types

Name	Storage Size	Description	Scope
smallint	2 bytes	Small range of integers	-32768 to +32767
integer	4 bytes	Typical choice for integers	-2147483648 to +2147483647
bigint	8 bytes	Large range of integers	-9223372036854775808 to +9223372036854775807
decimal	Variable	User-specified precision, exact	Up to 131072 digits before the decimal point and 16383 digits after the decimal point
numeric	Variable	User-specified precision, exact	Up to 131072 digits before the decimal point and 16383 digits after the decimal point
real	4 bytes	Variable precision, inexact	6-digit decimal precision
double precision	8 bytes	Variable precision, inexact	15-digit decimal precision
smallserial	2 bytes	Auto-incrementing small integer	1-32767
serial	4 bytes	Auto-incrementing integer	1-2147483647

bigserial	8 bytes	Auto-incrementing large integer	1-9223372036854775807

Character type

Name	Description
character varying(n), varchar(n)	Restricted variable length
character(n), char(n)	Fixed length, space-padded
text	1G

Binary data type

Name	Storage Size	Description
bytea	1 or 4 bytes plus actual binary string	Variable length binary string

Date Type

Name	Storage Size	Description	Minimum value	Maximum value	Resolution
timestamp [(p)] [without time zone]	8 bytes	Includes Date and Time (no Timezone)	4713 BC	294276 AD	1 microsecond / 14 bits
timestamp [(p)] with time zone	8 bytes	Includes Date and Time, with Timezone	4713 BC	294276 AD	1 microsecond / 14 bits
date	4 bytes	Date (without Time of Day)	4713 BC	5874897 AD	1 day
time [(p)] [without time zone]	8 bytes	Time of Day (no Date)	0:00:00	24:00:00	1 microsecond / 14 bits

time [(p)] with time zone	12 bytes	Time of Day only, with Timezone	00:00:00+1459	24:00:00-1459	1 microsecond / 14 bits
interval [fields] [(p)]	16 bytes	Time Interval	-178,000,000 years	178,000,000 years	1 microsecond / 14 bits

Boolean

Name	Storage Bytes	Description
boolean	1 byte	Status is true or false

For more data type introductions

Please click the link https://www.postgresql.org/docs/10/static/datatype.html to view more data type descriptions

Usage of auto-increment columns and sequences

Last updated: 2024-08-22 16:57:14

Creation and accessing of sequences

```
[tbase@VM_0_29_centos tbase_mgr]$ psql -p 15432 -U tbase -d postgres
psql (PostgreSQL 10 (TBase 2.01))
Type "help" for help.

-- Creating sequence
postgres=# create sequence tbase_seq;
CREATE SEQUENCE

-- Create a sequence only if it does not exist
postgres=# create sequence IF NOT EXISTS tbase_seq;
NOTICE: relation "tbase_seq" already exists, skipping
CREATE SEQUENCE

-- Viewing current sequence usage
postgres=# \\x
Expanded display is on.
postgres=# select * from tbase_seq ;
-[ RECORD 1 ]-
last_value | 1
log_cnt | 0
is_called | f

-- Getting next value in sequence
postgres=# select nextval('tbase_seq');
 nextval
---------
 1

-- Getting the current value of the sequence, which can only be used
after accessing nextval()
postgres=# select currval('tbase_seq');
 currval
---------
 1
```

```
-- You can also use the following method to get the currently used value
in the sequence
postgres=# select last_value from tbase_seq ;
 last_value
------------
 3

-- Setting current sequence value
postgres=# select setval('tbase_seq',1);
 setval
--------
 1
```

Using sequences in DML

```
-- Insert data, using sequence
postgres=# INSERT INTO t (id, nickname) VALUES (nextval('tbase_seq'),
'TBase is good');
INSERT 0 1

postgres=# select * from t;
 id | nickname
----+-----------
 1 | Tencent TBase
 2 | TBase is good
(2 rows)
```

Using sequence as default field value

```
-- Set sequence as default field value
postgres=# alter table t alter column id set default
nextval('tbase_seq');

-- Insert data, field uses sequence default value
postgres=# INSERT INTO t (nickname) VALUES ('hello TBase');

postgres=# select * from t;
 id | nickname
----+-------------
 3 | hello TBase
 1 | Tencent TBase
```

```
 2 | TBase is good
(3 rows)
```

Using sequence as field type

```
-- Drop table
postgres=# drop table t;
DROP TABLE

-- Create table, field uses sequence type
postgres=# create table t (id serial not null, nickname text);
CREATE TABLE

-- Insert data, field automatically uses sequence
postgres=# INSERT INTO t (nickname) VALUES ('hello TBase');
INSERT 0 1

postgres=# select * from t;
 id | nickname
----+-------------
 1 | hello TBase
(1 row)
```

Deleting sequence

```
-- Delete sequence
postgres=# drop sequence tbase_seq;
DROP SEQUENCE

-- Delete sequence (skip if it does not exist)
postgres=# drop sequence IF EXISTS tbase_seq;
NOTICE: sequence "tbase_seq" does not exist, skipping
DROP SEQUENCE
```

DDL operations for schema/table/index/view/materialized view, etc

Last updated: 2024-08-22 16:59:15

Database management

Create a database

To create a database, you must be a superuser or have the special CREATEDB privilege. By default, a new database is created by cloning the standard system database template1. You can specify a different template by writing TEMPLATE name. Specifically, by writing TEMPLATE template0, you can create a clean database that contains only the standard objects predefined by your TBase version.

Creating database by using default parameters

```
postgres=# create database tbase_db;
CREATE DATABASE
```

Specifying database to be cloned

```
postgres=# create database tbase_db_template TEMPLATE template0;
CREATE DATABASE
```

Specifying owner

```
postgres=# create role pgxz with login;
CREATE ROLE
postgres=# create database tbase_db_owner owner pgxz;
CREATE DATABASE
```

List databases and their details:

```
List of databases
| Name              | Owner | Encoding | Collate   | Ctype   | Access
privileges | Size   | Tablespace | Description      |
```

```
|-----------------|-------|----------|----------|--------|--------------
----|-------|------------|--------------------|
| tbase_db_owner    | pgxz   | UTF8      | en_US.utf8 | en_US.utf8 |
| 18 MB | pg_default |
```

Specifying encoding

```
postgres=# create database tbase_db_encoding ENCODING UTF8;
CREATE DATABASE
```

List databases and their details:

```
List of databases
| Name                    | Owner    | Encoding | Collate   | Ctype   |
Access privileges | Size   | Tablespace | Description       |
|----------------------|--------|----------|----------|--------|--------
----------|-------|------------|--------------------|
| tbase_db_encoding     | tbase   | UTF8      | en_US.utf8 | en_US.utf8 |
| 18 MB | pg_default |
```

Create a GBK encoded database

```
postgres=# CREATE DATABASE db_gbk template template0 encoding = gbk
LC_COLLATE = 'zh_CN.gbk' LC_CTYPE = 'zh_CN.gbk';
CREATE DATABASE
```

List databases and their details:

```
List of databases
| Name   | Owner | Encoding | Collate   | Ctype   | Access privileges |
Size   | Tablespace | Description       |
|-------|-------|----------|----------|--------|-------------------|-----
--|------------|--------------------|
| db_gbk | tbase | GBK       | zh_CN.gbk | zh_CN.gbk |                   |
19 MB | pg_default |
```

Create a GB18030 encoded database

```
postgres=# create database db_gb18030 template=template0
encoding=gb18030 LC_COLLATE = 'zh_CN.gb18030' LC_CTYPE =
```

```
'zh_CN.gb18030';
CREATE DATABASE
```

List databases and their details:

```
List of databases
| Name         | Owner | Encoding | Collate         | Ctype           |
Access privileges |
|--------------|-------|----------|-----------------|-----------------|------
------------|
| db_gb18030   | tbase | GB18030  | zh_CN.gb18030   | zh_CN.gb18030   |
...
```

Specifying sorting rule

```
postgres=# create database tbase_db_lc_collate lc_collate 'C';
CREATE DATABASE
```

List databases and their details:

```
List of databases
| Name                    | Owner | Encoding | Collate | Ctype | Access
privileges | Size   | Tablespace | Description      |
|-----------------------|-------|----------|---------|-------|-----------
--------|-------|------------|------------------|
| tbase_db_lc_collate   | tbase | UTF8     | C       | en_US.utf8 |
| 18 MB | pg_default |
```

Specifying grouping rule

```
postgres=# create database tbase_db_lc_ctype LC_CTYPE 'C' ;
CREATE DATABASE
```

List databases and their details:

```
List of databases
| Name                 | Owner | Encoding | Collate   | Ctype | Access
privileges | Size   | Tablespace | Description      |
|----------------------|-------|----------|-----------|-------|------------
------|-------|------------|------------------|
```

```
| tbase_db_lc_ctype | tbase | UTF8     | en_US.utf8 | C     |
| 18 MB | pg_default |
```

Configuring data connectivity

```
postgres=# create database tbase_db_allow_connections ALLOW_CONNECTIONS
true;
CREATE DATABASE
```

View database connection configuration:

```
postgres=# select datallowconn from pg_database where
datname='tbase_db_allow_connections';
datallowconn
--------------
t
```

Configuring the number of connections

```
postgres=# create database tbase_db_connlimit CONNECTION LIMIT 100;
CREATE DATABASE
```

View database connection number configuration:

```
postgres=# select datconnlimit from pg_database where
datname='tbase_db_connlimit';
datconnlimit
--------------
100
```

Configuring database replicability

```
postgres=# create database tbase_db_istemplate is_template true;
CREATE DATABASE
```

View database template configuration:

```
postgres=# select datistemplate from pg_database where
datname='tbase_db_istemplate';
```

```
datistemplate
---------------
t
```

Configuring multiple parameters together

```
postgres=# create database tbase_db_mul owner pgxz CONNECTION LIMIT 50
template template0 encoding 'utf8' lc_collate 'C';
CREATE DATABASE
```

Modify database configuration

Renaming database

```
postgres=# alter database tbase_db rename to tbase_db_new;
ALTER DATABASE
```

Modifying the number of connections

```
postgres=# alter database tbase_db_new connection limit 50;
ALTER DATABASE
```

Changing database owner

```
postgres=# alter database tbase_db_new owner to tbase;
ALTER DATABASE
```

Configuring default database running parameters

```
postgres=# alter database tbase_db_new set search_path to
public,pg_catalog,pg_oracle;
ALTER DATABASE
```

More usage of `SET` can be found in the operation and maintenance documentation.

Unsupported features of Alter database

Item	Remarks

encoding	Encoding
lc_collate	Sorting rule
lc_ctype	Grouping rule

Dropping a Database

Dropping a Database

```
postgres=# drop database tbase_db_new;
DROP DATABASE
```

If you drop a database when a session is connected to it, the following error will be reported:

```
postgres=# drop database tbase_db_template;
ERROR:  database "tbase_db_template" is being accessed by other users
DETAIL:  There is 1 other session using the database.
```

You can use the following method to disconnect the session and then drop the database:

```
postgres=# select pg_terminate_backend(pid) from pg_stat_activity where
datname='tbase_db_template';
 pg_terminate_backend
----------------------
 t
(1 row)
```

Retry deleting the database:

```
postgres=# drop database tbase_db_template;
DROP DATABASE
```

```
Please note that tables in Markdown format may not display vertical
borders when rendering, but the above content conforms to Markdown
syntax rules. If further adjustments are needed, please let me know.
```

Schema Management

A schema is essentially a namespace, known by different names in different database systems. In Oracle, it is usually called a user. In SQL Server, it is called a framework. In MySQL, it is called a database. A schema may contain tables, data types, functions, and operators. Object names can be duplicated in different schemas. To access objects in a specific schema, you can use the format "SchemaName.ObjectName".

Creating Schema

- Standard statement:

```
postgres=# create schema tbase;
  CREATE SCHEMA
```

- Use the extended syntax to create a schema only if it does not exist:

```
postgres=# create schema if not exists tbase;
NOTICE: schema "tbase" already exists, skipping
CREATE SCHEMA
```

- Specify the owner of the schema:

```
postgres=# create schema tbase_pgxz AUTHORIZATION pgxz;
CREATE SCHEMA
```

Use the `\dn` command to list schemas and their owners:

```
List of schemas
Name | Owner
------------+-------
tbase_pgxz | pgxz
(1 row)
```

Modifying schema attributes

- Modify the schema name:

```
postgres=# alter schema tbase rename to tbase_new;
ALTER SCHEMA
```

- Change the owner:

```
postgres=# alter schema tbase_pgxz owner to tbase;
ALTER SCHEMA
```

Deletion Mode

- Schema deletion command:

```
postgres=# drop schema tbase_new;
DROP SCHEMA
```

- If there are objects in the schema, dropping will fail, and error information will be prompted. You can use the `CASCADE` option to force delete the schema and its dependent objects:

```
postgres=# drop schema tbase_pgxz CASCADE;
NOTICE: drop cascades to table tbase_pgxz.t
DROP SCHEMA
```

Configuring User Access to Schema

A regular user needs two steps for authorization to access an object in a schema: access to the object itself and access to the schema.

- Authorize users to access a specific table:

```
postgres=# grant select on tbase.t to pgxz;
GRANT
```

- Authorize users to use a specific schema:

```
postgres=# grant USAGE on SCHEMA tbase to pgxz;
GRANT
```

Configuring Schema Access Order

The TBase Database uses `search_path` run variable to configure the access order of data objects.

- Search path of the currently connected user:

```
postgres=# show search_path ;
```

```
search_path
-----------------
"$user", public
(1 row)
```

- When creating a data table, if the schema is not specified, the table will be stored under the first search schema:

```
postgres=# create table t(id int,mc text);
NOTICE: Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
```

- When accessing an object not in the search path, you need to enter its full path:

```
postgres=# select * from t1;
ERROR: relation "t1" does not exist
postgres=# select * from tbase_schema.t1;
id | mc
----+----
(0 rows)
```

Creating and Deleting Data Tables

Creating tables without specifying a shard key

When creating tables without specifying a shard key, the system defaults to using the first field as the shard key.

```
postgres=# create table t_first_col_share(id serial not null, nickname
text);
CREATE TABLE
postgres=# \d+ t_first_col_share
```

Column	Type	Modifiers	Storage	Stats target	Description
id	integer	not null default nextval('t_first_col_share_id_seq'::regclass)	plain		

nickname	text		extended		

Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES

Principles for Shard Key Selection:

- Only one field can be selected as the shard key.
- If there is a primary key, the primary key should be selected as the shard key.
- If the primary key is a composite field, select the field with the most distinct values as the shard key.
- Composite fields can also be concatenated into a new field to serve as the shard key.
- If there is no primary key, UUID can be used as the shard key.
- In short, ensure the data is distributed as evenly as possible.

Creating tables by specifying a shard key

When creating tables with a specified shard key, you can use the `distribute by shard(column_name)` statement to specify it.

```
postgres=# create table t_appoint_col(id serial not null, nickname text)
distribute by shard(nickname);
CREATE TABLE
postgres=# \d+ t_appoint_col
```

Table "public.t_appoint_col"

Column	Type	Modifiers	Storage	Stats target	Description
id	integer	not null default nextval('t_appoint_col_id_seq'::regclass)	plain		
nickname	text		extended		

Has OIDs: no
Distribute By SHARD(nickname)
Location Nodes: ALL DATANODES

Creating tables by specifying a group

Creating tables by specifying a group uses the `to group group_name` statement to specify it.

```
postgres=# create table t (id integer, nc text) distribute by shard(id)
to group default_group;
CREATE TABLE
postgres=# \d+ t
```

Table "public.t"

Column	Type	Modifiers	Storage	Stats target	Description
id	integer		plain		
nc	text		extended		

Has OIDs: no
Distribute By SHARD(id)
Location Nodes: ALL DATANODES

Logical partitioned table

Range partitioned table

Creating primary partitions and subpartitions.

```
postgres=# create table t_native_range (f1 bigint, f2 timestamp default
now(), f3 integer) partition by range(f2) distribute by shard(f1) to
group default_group;
NOTICE: Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
```

Creating two subtables:

```
postgres=# create table t_native_range_201709 partition of
t_native_range (f1, f2, f3) for values from ('2017-09-01') to ('2017-10-
01');
postgres=# create table t_native_range_201710 partition of
t_native_range (f1, f2, f3) for values from ('2017-10-01') to ('2017-11-
01');
```

Note that adding subpartitions will affect the DML operations on the main table data.

Default partitioned table

An error will occur if data is inserted without a default partitioned table.

```
postgres=# insert into t_native_range values(2,'2017-08-01',2);
ERROR: no partition of relation "t_native_range" found for row
```

Adding a default partitioned table:

```
postgres=# CREATE TABLE t_native_range_default PARTITION OF
t_native_range DEFAULT;
CREATE TABLE
postgres=# insert into t_native_range values(2,'2017-08-01',2);
INSERT 0 1
```

MAXVALUE partition

Creating a MAXVALUE partition.

```
postgres=# CREATE TABLE t_native_range_maxvalue PARTITION OF
t_native_range for values from ('2017-11-01') to (maxvalue);
CREATE TABLE
postgres=# insert into t_native_range values(1,'2017-11-01',1);
INSERT 0 1
```

All data greater than `2017-11-01` will be stored in the subtable `t_native_range_maxvalue`.

MINVALUE partition

Creating a MINVALUE partition.

```
postgres=# CREATE TABLE t_native_range_minvalue PARTITION OF
t_native_range for values from (minvalue) to ('2017-09-01');
CREATE TABLE
postgres=# insert into t_native_range values(1,'2017-08-01',1);
INSERT 0 1
```

View table structure

View the table structure of `t_native_range`.

```
postgres=# \d+ t_native_range
```

Table "tbase_pg_proc.t_native_range"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
f1	bigint				plain		
f2	timestamp without time zone			now()	plain		
f3	integer				plain		

Partition key: RANGE (f2)
Partitions: ...
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES

List partitioned table

Creating primary partitions and subpartitions.

```
postgres=# create table t_native_list(f1 bigserial not null, f2 text, f3
integer, f4 date) partition by list(f2) distribute by shard(f1) to group
default_group;
```

Create two subtables, storing "Guangdong" and "Beijing" respectively.

```
postgres=# create table t_list_gd partition of t_native_list(f1, f2, f3,
f4) for values in ('Guangdong');
postgres=# create table t_list_bj partition of t_native_list(f1, f2, f3,
f4) for values in (' Beijing ');
```

View the table structure:

```
postgres=# \d+ t_native_list
```

Table "tbase.t_native_list"

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
f1	bigint		not null	nextval('t_native_list_f1_seq'::regclass)	plain		
f2	text				extended		
f3	integer				plain		
f4	date				plain		

Partition key: LIST (f2)
Partitions: ...
Distribute By: SHARD(f1)
Location Nodes: ALL DATANODES

Creating a default partition

An error will occur if there is no default partition, and data insertion will fail.

```
postgres=# insert into t_native_list values(1,' Shanghai
',1,current_date);
ERROR: no partition of relation "t_native_list" found for row
```

After creation, insertion can proceed normally.

```
postgres=# CREATE TABLE t_native_list_default PARTITION OF t_native_list
DEFAULT;
CREATE TABLE
postgres=# insert into t_native_list values(1,' Shanghai
',1,current_date);
INSERT 0 1
```

Hash partitioned table

When creating a hash table with 4 subpartitions, the number of partitions must be specified. The partition number is used as an operator to calculate the partition of each row of data.

Currently, hash partitioning does not support add and delete operations.

```
postgres=# create table t_hash_partition(f1 int, f2 int) partition by
hash(f2);
create table t_hash_partition_1 partition of t_hash_partition FOR VALUES
WITH(MODULUS 4, REMAINDER 0);
create table t_hash_partition_2 partition of t_hash_partition FOR VALUES
WITH(MODULUS 4, REMAINDER 1);
create table t_hash_partition_3 partition of t_hash_partition FOR VALUES
WITH(MODULUS 4, REMAINDER 2);
create table t_hash_partition_4 partition of t_hash_partition FOR VALUES
WITH(MODULUS 4, REMAINDER 3);
```

Insert data:

```
postgres=# insert into t_hash_partition values(1,1),(2,2),(3,3);
COPY 3
```

Query result:

```
postgres=# select * from t_hash_partition;
+----+----+
| f1 | f2 |
+----+----+
|  1 |  1 |
|  3 |  3 |
|  2 |  2 |
+----+----+
(3 rows)
```

TBase automatically performs pruning based on partition values:

```
postgres=# explain select * from t_hash_partition where f2=2;
                                  QUERY PLAN
------------------------------------------------------------------------------
-------
 Remote Fast Query Execution  (cost=0.00..0.00 rows=0 width=0)
   Node/s: dn001, dn002
   ->  Append  (cost=0.00..26.88 rows=7 width=8)
         ->  Seq Scan on t_hash_partition_3  (cost=0.00..26.88 rows=7
width=8)
```

```
                Filter: (f2 = 2)
(5 rows)
```

Multi-level partitioned table

Create Main Table

```
postgres=# create table t_native_mul_list(f1 bigserial not null, f2
integer, f3 text, f4 text, f5 date)
partition by list (f3) distribute by shard(f1) to group default_group;
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
```

Create Secondary Table

```
postgres=# create table t_native_mul_list_gd partition of
t_native_mul_list for values in ('Guangdong')
partition by range(f5);
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE

postgres=# create table t_native_mul_list_bj partition of
t_native_mul_list for values in (' Beijing ')
partition by range(f5);
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE

postgres=# create table t_native_mul_list_sh partition of
t_native_mul_list for values in (' Shanghai ');
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
```

Create Tertiary Table

```
postgres=# create table t_native_mul_list_gd_201701 partition of
t_native_mul_list_gd(f1,f2,f3,f4,f5)
for values from ('2017-01-01') to ('2017-02-01');
```

```
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE

postgres=# create table t_native_mul_list_gd_201702 partition of
t_native_mul_list_gd(f1,f2,f3,f4,f5)
for values from ('2017-02-01') to ('2017-03-01');
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE

postgres=# create table t_native_mul_list_bj_201701 partition of
t_native_mul_list_bj(f1,f2,f3,f4,f5)
for values from ('2017-01-01') to ('2017-02-01');
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE

postgres=# create table t_native_mul_list_bj_201702 partition of
t_native_mul_list_bj(f1,f2,f3,f4,f5)
for values from ('2017-02-01') to ('2017-03-01');
NOTICE:  Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
```

TBase supports mixed use of level 1 and level 2 partitions, partitions do not need to be equal level.

Database Replication Table

Replication tables store the same data on all dn nodes. After creating a replication table, you can insert data and query the same results on different nodes. Replication tables do not support triggers and foreign keys.

Create Replication Table

```
CREATE TABLE t_rep (id INT, mc TEXT) DISTRIBUTE BY REPLICATION TO GROUP
default_group;
```

Inserting Data

```
INSERT INTO t_rep VALUES(1, 'TBase'), (2, 'pgxz');
```

Querying Data

```
EXECUTE DIRECT ON (dn001) 'SELECT * FROM t_rep';
EXECUTE DIRECT ON (dn002) 'SELECT * FROM t_rep';
```

Columnar Storage Table Management

Columnar storage tables are only supported in TBase-v3.

Create Columnar Storage Table

```
CREATE TABLE t_col_test (f1 INT, f2 VARCHAR(32), f3 DATE) WITH
(orientation='column');
```

View Columnar Storage Table Structure

```
\d+ t_col_test
```

Specify Compression Type

- Use a uniform compression level for all columns:

```
CREATE TABLE t_col1 (f1 INT, f2 INT, f3 DATE) WITH
(orientation=column, compression=HIGH);
```

- Configure Default Compression Level:

```
SHOW default_rel_compression;
SET default_rel_compression TO 'middle';
```

- Set different compression levels for different columns:

```
CREATE TABLE t_col3 (
  f1 INT ENCODING(compression=no),
  f2 INT ENCODING(compression=low),
  f3 INT ENCODING(compression=middle),
  f4 INT ENCODING(compression=high)
) WITH (orientation=column);
```

Columnar partitioned table

Create Main Table

```
CREATE TABLE t_native_range (
  f1 BIGINT,
  f2 TIMESTAMP DEFAULT NOW(),
  f3 INTEGER
) PARTITION BY RANGE (f2) WITH (orientation=column, compression=low);
```

Create Subtable

```
CREATE TABLE t_native_range_201709 PARTITION OF t_native_range FOR
VALUES FROM ('2017-09-01') TO ('2017-10-01') WITH (orientation=column,
compression=low);
CREATE TABLE t_native_range_201710 PARTITION OF t_native_range FOR
VALUES FROM ('2017-10-01') TO ('2017-11-01') WITH (orientation=column,
compression=low);
```

View table structure

```
\d+ t_native_range
```

Columnar Stash Table

The Stash feature for columnar tables allows individual insert or update operation data to be initially stored in the Stash, then persistently stored in the columnar storage structure as per the settings.

Create Stash Table

```
CREATE TABLE t_stash (f1 INT, f2 INT) WITH (orientation=column,
stash_enabled=on);
```

Modify Table to Stash Table

```
ALTER TABLE t_stash SET (stash_enabled=off);
```

Foreign Table Management

Create hdfs_fdw Plugin

```
CREATE EXTENSION hdfs_fdw;
```

Create Server

Create COS Server

```
CREATE SERVER $cos_server_name
 FOREIGN DATA WRAPPER hdfs_fdw
 OPTIONS (
  address 'cos://$bucketname',
  appid '$appid',
  access_keyid '$ak',
  secret_accesskey '$sk',
  region '$region',
  client_type 'cos'
);
```

Create HDFS Server

```
CREATE SERVER $hdfs_server_name
 FOREIGN DATA WRAPPER hdfs_fdw
 OPTIONS (
  address 'ofs://xxxxxx.chdfs.ap-guangzhou.myqcloud.com',
  appid '$appid',
  client_type 'hdfs'
);
```

Create Fusion Bucket Server

```
CREATE SERVER $cosn_server_name
 FOREIGN DATA WRAPPER hdfs_fdw
 OPTIONS (
  address 'cosn://$bucketname',
  appid '$appid',
  access_keyid '$ak',
  secret_accesskey '$sk',
  region '$region',
  client_type 'cosn'
```

```
);
```

Create Postgres Server

```
CREATE SERVER postgres_fdw_postgres_db
 FOREIGN DATA WRAPPER postgres_fdw
 OPTIONS (
  host '172.16.64.14',
  port '11345',
  dbname 'postgres'
);
```

Configure User Mapping

```
CREATE USER MAPPING FOR $TDSQL-A_USER SERVER $cos_server_name;
CREATE USER MAPPING FOR $TDSQL-A_USER SERVER $hdfs_server_name;
CREATE USER MAPPING FOR $TDSQL-A_USER SERVER $cosn_server_name;
```

Create External Table

Create Non-partitioned Table

```
CREATE FOREIGN TABLE test_csv(
  id INT,
  name TEXT
)
SERVER $ServerName
OPTIONS (
  FORMAT 'csv',
  FOLDERNAME '$data directory/',
  distribute 'shard'
);
```

Create Table in TEXT Format

```
CREATE FOREIGN TABLE test_text(
  id INT,
  name TEXT
)
SERVER $ServerName
OPTIONS (
```

```
  FORMAT 'text',
  DELIMITER '$Column Separator',
  FOLDERNAME '$data directory/',
  distribute 'shard'
);
```

Create Table in ORC Format

```
CREATE FOREIGN TABLE orc_table(
  a BIGINT,
  b TEXT,
  c FLOAT
)
SERVER $ServerName
OPTIONS (
  FORMAT 'orc',
  FOLDERNAME '$data directory/',
  distribute 'shard'
);
```

Creating a Partition Table

```
CREATE FOREIGN TABLE login_logs_parquet(
  l_id TEXT,
  l_loginName TEXT,
  l_date TEXT,
  year TEXT,
  month TEXT
)
SERVER cosn_server
OPTIONS (
  FORMAT 'parquet',
  FOLDERNAME '$data directory/',
  distribute 'shard',
  PARTITION 'year,month'
);
```

Query the external table and import data into the internal table

```
INSERT INTO Internal Table SELECT * FROM External Table;
```

dblink External Table

Create dblink

```
CREATE DATABASE LINK abc
CONNECT TO "TBASE"
IDENTIFIED BY 'tbase2018'
USING '(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.0.17)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = ora12c)
  )
)';
```

Query dblink System Table

```
SELECT * FROM pg_dblink;
```

Access Remote Table

```
SELECT * FROM t1@abc;
```

Drop dblink

```
DROP DATABASE LINK abc;
```

tbase_dblink

Create postgres_fdw

```
CREATE EXTENSION postgres_fdw;
```

Create dblink

```
SET dblink_types = 'postgresql';
CREATE DATABASE LINK tbase_dblink
CONNECT TO "tbase"
IDENTIFIED BY 'tbase@2017'
```

```
USING '(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 172.16.0.23)(PORT = 11379))
  (CONNECT_DATA =(SERVER = DEDICATED)(SERVICE_NAME = postgres)
(SCHEMA_NAME = public))
  (COLLATE = YES)(DEFAULT = YES)(NOT_NULL = YES)
)';
```

Query dblink System Table

```
SELECT * FROM pg_dblink WHERE dblinkname='tbase_dblink';
```

Access Remote Table

```
SELECT * FROM t2@tbase_dblink;
```

Drop dblink

```
DROP DATABASE LINK tbase_dblink;
```

Create table using IF NOT EXISTS

```
CREATE TABLE IF NOT EXISTS t(id INT, mc TEXT);
```

Create table with designated mode

```
CREATE TABLE public.t(id INT, mc TEXT);
```

Create data table using query results

```
CREATE TABLE t(id INT, mc TEXT) DISTRIBUTE BY SHARD(mc);
INSERT INTO t VALUES(1, 'tbase');
CREATE TABLE t_as AS SELECT * FROM t;
```

Dropping a Table

```
DROP TABLE t;
DROP TABLE public.t;
DROP TABLE IF EXISTS t;
```

```
DROP TABLE tbase_schema.t1 CASCADE;
```

Drop partitioned subtable

```
DROP TABLE t_time_range_part_1;
```

Add partitioned subtable

Add partitioned subtable at the end

```
ALTER TABLE t1_pt ADD PARTITIONS 2;
```

Add partitioned subtable in the middle

```
CREATE TABLE t_time_range_part_1 (
  f1 BIGINT NOT NULL,
  f2 TIMESTAMP WITHOUT TIME ZONE,
  f3 BIGINT
);
ALTER TABLE t_time_range_part_1 ADD CONSTRAINT t_time_range_pkey_part_1
PRIMARY KEY (f1);
CREATE INDEX t_time_range_f2_idx_part_1 ON t_time_range_part_1(f2);
INSERT INTO t_time_range VALUES(1, '2017-10-1', 1);
SELECT * FROM t_time_range;
```

Create and drop index

Regular Index

```
postgres=# create index t_appoint_id_idx on t_appoint_col using
btree(id);
CREATE INDEX
```

Unique index

Creating unique index

```

```

```
postgres=# create unique index t_first_col_share_id_uidx on
t_first_col_share using btree(id);
CREATE INDEX
```

You cannot create a unique index for non-shard key fields

```
postgres=# create unique index t_first_col_share_nickname_uidx on
t_first_col_share using btree(nickname);
```

Error message:

```
ERROR: Unique index of partitioned table must contain the hash/modulo
distribution column.
```

Index on expression

Create table t_upper

```
postgres=# create table t_upper(id int,mc text);
```

Create index t_upper_mc

```
postgres=# create index t_upper_mc on t_upper(mc);
```

Insert data and analyze

```
postgres=# insert into t_upper select t,md5(t::text) from
generate_series(1,10000) as t;
postgres=# analyze t_upper;
```

Query plan, use expression index.

```
postgres=# explain select * from t_upper where upper(mc)=md5('1');
```

Conditional index

Create table t_sex

```
postgres=# create table t_sex(id int,sex text);
```

Create index t_sex_sex_idx

```
postgres=# create index t_sex_sex_idx on t_sex (sex);
```

Insert data and analyze

```
postgres=# insert into t_sex select t,'男' from
generate_series(1,1000000) as t;
postgres=# insert into t_sex select t,'女' from generate_series(1,100) as
t;
postgres=# analyze t_sex;
```

Query plan, use conditional index.

```
postgres=# explain select * from t_sex where sex ='女';
```

gist index

Create table t_trgm and create gist index

```
postgres=# create table t_trgm (id int,trgm text,no_trgm text);
postgres=# create index t_trgm_trgm_idx on t_trgm using gist(trgm
gist_trgm_ops);
```

Note: Only row-store table is supported

gin index

pg_trgm index

Delete index t_trgm_trgm_idx

```
postgres=# drop index t_trgm_trgm_idx;
```

Create GIN index t_trgm_trgm_idx

```
postgres=# create index t_trgm_trgm_idx on t_trgm using gin(trgm
```

```
gin_trgm_ops);
```

Note: Only row-store table is supported

JSONB index

Create table t_jsonb and create JSONB index

```
postgres=# create table t_jsonb(id int,f_jsonb jsonb);
postgres=# create index t_jsonb_f_jsonb_idx on t_jsonb using
gin(f_jsonb);
```

Array index

Create table t_array and insert data

```
postgres=# create table t_array(id int, mc text[]);
postgres=# insert into t_array select t,('{'||md5(t::text)||'}')::text[]
from generate_series(1,1000000) as t;
```

Create array index t_array_mc_idx

```
postgres=# create index t_array_mc_idx on t_array using gin(mc);
```

Btree_GIN any field index

Create table gin_mul and insert data

```
postgres=# create table gin_mul(f1 int, f2 int, f3 timestamp, f4 text,
f5 numeric, f6 text);
postgres=# insert into gin_mul select random()*5000, random()*6000,
now()+((3000060000*random())||' sec')::interval , md5(random()::text),
round((random()*100000)::numeric,2), md5(random()::text) from
generate_series(1,1000000);
```

Create btree_gin extension and create index gin_mul_gin_idx

```
postgres=# create extension btree_gin;
postgres=# create index gin_mul_gin_idx on gin_mul using
gin(f1,f2,f3,f4,f5,f6);
```

Multi-Field index

Create an index that includes multiple fields to improve query performance, especially when using `OR` query conditions. However, note that Bitmap scanning supports conditions on at most two different fields.

```
CREATE TABLE t_mul_idx (f1 int, f2 int, f3 int, f4 int);
CREATE INDEX t_mul_idx_idx ON t_mul_idx(f1, f2, f3);
```

Precautions for using multi-field index

- When using `OR` query conditions, Bitmap scanning supports conditions on at most two different fields.

Sample query plan

- Query using multi-field index:

```
EXPLAIN SELECT * FROM t_mul_idx WHERE f1=1 OR f2=2;
```

- When the query condition exceeds two fields, Seq Scan will be used instead of Bitmap Heap Scan:

```
EXPLAIN SELECT * FROM t_mul_idx WHERE f1=1 OR f2=2 OR f3=3;
```

Index and Performance

- If all the return fields of the query are in the index, an Index Only Scan can be implemented, reducing I/O overhead.
- In insert operations, multi-field indexing usually performs better than single-field indexing.

Index > Global index

Global Index (Global Index) is used to resolve data localization issues in MPP distributed databases, especially in the absence of a scatter key.

Supported Features

- General shard table global index: Version >= 5.06.2
- Partition table global index: Version >= 5.06.3
- Distributed global index: Version >= 5.06.3

Create Global Index

```
CREATE TABLE t1 (f1 int, f2 int);
CREATE GLOBAL INDEX t1_f2_idx ON t1(f2);
```

View Index Type

```
\d+ t1
```

Global Index Execution Plan

```
EXPLAIN SELECT * FROM t1 WHERE f2=1;
```

Global Unique Index

```
CREATE GLOBAL UNIQUE INDEX t1_f2_gidx ON t1(f2);
```

Features Not Supported by Global Index

- The `truncate` operation is not supported.
- The `reindex` of global indexes is not supported.
- Creating global indexes `CONCURRENTLY` is not supported.
- Executing `vacuum full` on tables with global indexes is not supported.

Deleting an Index

```
DROP INDEX t_appoint_id_idx;
```

Modifying table structure

Modifying table name

```
postgres=# ALTER TABLE t RENAME TO tbase;
```

Add comments to tables or fields

```
postgres=# COMMENT ON TABLE tbase IS 'TBase distributed relational
database system';
postgres=# COMMENT ON COLUMN tbase.nickname IS 'TBase nickname is
Elephant';
```

Add fields to table

```
postgres=# ALTER TABLE tbase ADD COLUMN age INTEGER;
```

Modify field type

```
postgres=# ALTER TABLE tbase ALTER COLUMN age TYPE FLOAT8;
```

Modify field default value

```
postgres=# ALTER TABLE tbase ALTER COLUMN age SET DEFAULT 0.0;
```

Delete field

```
postgres=# ALTER TABLE tbase DROP COLUMN age;
```

Add primary key

```
postgres=# ALTER TABLE t ADD CONSTRAINT t_id_pkey PRIMARY KEY (id);
```

Delete primary key

```
postgres=# ALTER TABLE t DROP CONSTRAINT t_id_pkey;
```

Rebuild primary key

```
postgres=# CREATE UNIQUE INDEX CONCURRENTLY t_id_temp_idx ON t (id);
postgres=# ALTER TABLE t DROP CONSTRAINT t_pkey, ADD CONSTRAINT t_pkey
PRIMARY KEY USING INDEX t_id_temp_idx;
```

Add foreign key

```
CREATE TABLE t_p (f1 INT NOT NULL, f2 INT, PRIMARY KEY (f1));
CREATE TABLE t_f (f1 INT NOT NULL, f2 INT);
postgres=# ALTER TABLE t_f ADD CONSTRAINT t_f_f1_fkey FOREIGN KEY (f1)
REFERENCES t_p (f1);
```

Delete foreign key

```
postgres=# ALTER TABLE t_f DROP CONSTRAINT t_f_f1_fkey;
```

Modify the schema of the table

```
postgres=# ALTER TABLE t SET SCHEMA public;
```

Change the table owner

```
postgres=# ALTER TABLE tbase OWNER TO pgxz;
```

Change column name

```
postgres=# ALTER TABLE tbase RENAME COLUMN city TO cityname;
```

Modify table fill factor

```
postgres=# ALTER TABLE t1 SET (fillfactor=80);
```

Add a trigger

INSERT trigger

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_insert_trigger_func()
RETURNS TRIGGER AS $body$
BEGIN
  IF NEW.f2 < 0 THEN
    NEW.f2 := 0;
  END IF;
  RETURN NEW;
END;
```

```
$body$
LANGUAGE plpgsql;

postgres=# CREATE TRIGGER t_trigger_insert_trigger BEFORE INSERT ON
t_trigger FOR EACH ROW EXECUTE PROCEDURE
t_trigger_insert_trigger_func();
```

UPDATE trigger

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_update_trigger_func()
RETURNS TRIGGER AS $body$
BEGIN
  IF NEW.f2 < 0 THEN
    NEW.f2 := OLD.f2;
  END IF;
  RETURN NEW;
END;
$body$
LANGUAGE plpgsql;

postgres=# CREATE TRIGGER t_trigger_update_trigger BEFORE UPDATE ON
t_trigger FOR EACH ROW EXECUTE PROCEDURE
t_trigger_update_trigger_func();
```

DELETE trigger

```
postgres=# CREATE OR REPLACE FUNCTION t_trigger_delete_trigger_func()
RETURNS TRIGGER AS $body$
BEGIN
  IF OLD.f2 = 0 THEN
    RETURN NULL;
  END IF;
  RETURN OLD;
END;
$body$
LANGUAGE plpgsql;

postgres=# CREATE TRIGGER t_trigger_delete_trigger BEFORE DELETE ON
t_trigger FOR EACH ROW EXECUTE PROCEDURE
t_trigger_delete_trigger_func();
```

Multiple event triggers

```
postgres=# CREATE TABLE t_trigger_mulevent(f1 INT, f2 INT);
CREATE OR REPLACE FUNCTION t_trigger_mulevent_func() RETURNS TRIGGER AS
$body$
BEGIN
  IF NEW.f2 < 0 THEN
    NEW.f2 := 0;
  END IF;
  RETURN NEW;
END;
$body$
LANGUAGE plpgsql;

postgres=# CREATE TRIGGER t_trigger_insert_update_trigger BEFORE INSERT
OR UPDATE ON t_trigger_mulevent FOR EACH ROW EXECUTE PROCEDURE
t_trigger_mulevent_func();
```

Deleting Triggers

```
postgres=# DROP TRIGGER t_trigger_insert_update_trigger ON
t_trigger_mulevent;
postgres=# DROP FUNCTION t_trigger_mulevent_func();
```

Creating and dropping views

Creating View

```
postgres=# create view t_range_view as select * from t_range;
CREATE VIEW
postgres=# select * from t_range_view;
```

f1	f2	f3	f4
1	2017-09-27 23:17:39.674318	1	
2	2017-09-27 23:17:39.674318	50	
2	2017-09-27 23:17:39.674318	110	
1	2017-09-27 23:39:45.841093	151	

3	2017-09-27 23:17:39.674318	100	

(5 rows)

Redefining data type

```
postgres=# create view t_range_view as select f1,f2::date from t_range;
CREATE VIEW
postgres=# select * from t_range_view;
```

f1	f2
1	2017-09-27
2	2017-09-27
2	2017-09-27
1	2017-09-27
3	2017-09-27

(5 rows)

Redefine and rename data type.

```
postgres=# create view t_range_view as select f1,f2::date as mydate from
t_range;
CREATE VIEW
postgres=# select * from t_range_view;
```

f1	mydate
1	2017-09-27
2	2017-09-27
2	2017-09-27
1	2017-09-27
3	2017-09-27

(5 rows)

TBase supports synced renaming for tables or fields referenced in a view to avoid affecting data.

```
postgres=# \d+ t_view
View "tbase.t_view"
 Column | Type | Collation | Nullable | Default | Storage | Description
--------+------+-----------+----------+---------+---------+-------------
 id     | integer |           |          |         | plain   |
 mc     | text    |           |          |         | extended |
View definition:
 SELECT t.id,
 t.mc
 FROM t;

postgres=# alter table t rename to t_new;
ALTER TABLE
Time: 62.875 ms

postgres=# alter table t_new rename mc to mc_new;
ALTER TABLE
Time: 22.081 ms

postgres=# \d+ t_view
View "tbase.t_view"
 Column | Type | Collation | Nullable | Default | Storage | Description
--------+------+-----------+----------+---------+---------+-------------
 id     | integer |           |          |         | plain   |
 mc     | text    |           |          |         | extended |
View definition:
 SELECT t_new.id,
 t_new.mc_new AS mc
 FROM t_new;
```

SELECT Statement

Last updated: 2024-08-22 16:59:53

Access Function

```
postgres=# select md5(random()::text);
```

Result:

```
 md5
----------------------------------
 3eb6c0c8f8355f0b0f0cad7a8f0f7491
```

Data Sorting

Sort by a Column

```
postgres=# INSERT into tbase (nickname) VALUES('TBase Good');
postgres=# INSERT into tbase (id,nickname) VALUES(1,'The era of TBase
distributed database has arrived');
postgres=# select * from tbase order by id;
```

Result:

```
 id | nickname
----+-----------------------------
 1  | hello TBase
 1  | The era of TBase distributed database has arrived
 2  | TBase Good
(3 rows)
```

Sort by First Column

```
postgres=# select * from tbase order by 1;
```

Sort by id Ascending, then by nickname Descending

```
postgres=# select * from tbase order by id, nickname desc;
```

Random Sort

```
postgres=# select * from tbase order by random();
```

Calculate Sort

```
postgres=# select * from tbase order by md5(nickname);
```

Sort using Subquery

```
postgres=# select * from tbase order by (select id from tbase order by
random() limit 1);
```

NULL Value Sort Result Processing

```
postgres=# insert into tbase values(4,null);
postgres=# select * from tbase order by nickname nulls first;
postgres=# select * from tbase order by nickname nulls last;
```

Sort by Pinyin

```
postgres=# select * from (values ('张三'), ('李四'),('陈五')) t(myname)
order by myname;
postgres=# select * from (values ('张三'), ('李四'),('陈五')) t(myname)
order by convert(myname::bytea,'UTF-8','GBK');
postgres=# select * from (values ('张三'), ('李四'),('陈五')) t(myname)
order by convert_to(myname,'GBK');
postgres=# select * from (values ('张三'), ('李四'),('陈五')) t(myname)
order by myname collate "zh_CN.utf8";
```

WHERE Condition Usage

Single Condition Query

```
postgres=# select * from tbase where id=1;
```

Multiple Conditions AND

```
postgres=# select * from tbase where id=1 and nickname like '%h%';
```

Multiple Conditions OR

```
postgres=# select * from tbase where id=2 or nickname like '%h%';
```

ilike Case-insensitive Matching

```
postgres=# create table t_ilike(id int,mc text);
postgres=# insert into t_ilike values(1,'tbase'),(2,'TBase');
postgres=# select * from t_ilike where mc ilike '%tb%';
```

WHERE Condition Supports Subqueries

```
postgres=# select * from tbase where id=(select (random()*2)::integer
from tbase order by random() limit 1);
```

NULL Value Query Method

```
postgres=# select * from tbase where nickname is null;
postgres=# select * from tbase where nickname is not null;
```

exists, returns true as long as there is a record

```
postgres=# create table t_exists1(id int,mc text);
postgres=# insert into t_exists1 values(1,'tbase'),(2,'TBase');
postgres=# create table t_exists2(id int,mc text);
postgres=# insert into t_exists2 values(1,'tbase'),(1,'TBase');
postgres=# select * from t_exists1 where exists(select 1 from t_exists2
where t_exists1.id=t_exists2.id);
```

Equivalent EXISTS Writing

```
postgres=# select t_exists1.* from t_exists1, (select distinct id from
t_exists2) as t where t_exists1.id=t.id;
```

Paginated query

By default, starts from the first record and returns one record:

```
postgres=# select * from tbase limit 1;
```

Result:

```
 id | nickname
----+-------------
 1 | hello TBase
(1 row)
```

Use `offset` to specify the starting record. 0 indicates starting from the first record and returns 1 record:

```
postgres=# select * from tbase limit 1 offset 0;
```

Result:

```
 id | nickname
----+-------------
 1 | hello TBase
(1 row)
```

Starts from the 3rd record and returns two records:

```
postgres=# select * from tbase limit 1 offset 2;
```

Result:

```
 id | nickname
----+-----------------
 1 | The era of TBase Distributed Database has arrived
(1 row)
```

Using `order by` can obtain an ordered result:

```
postgres=# select * from tbase order by id limit 1 offset 2;
```

Result:

```
 id | nickname
----+-----------
 2 | TBase is good
(1 row)
```

Merge multiple query results

Do not filter duplicate records:

```
postgres=# select * from tbase union all select * from t_appoint_col;
```

Result:

```
 id | nickname
----+-----------------
 1 | hello TBase
 2 | TBase is good
 1 | The era of TBase Distributed Database has arrived
 1 | hello TBase
(4 rows)
```

Filter duplicate records:

```
postgres=# select * from tbase union select * from t_appoint_col;
```

Result:

```
 id | nickname
----+-----------------
 1 | The era of TBase Distributed Database has arrived
 1 | hello TBase
 2 | TBase is good
(3 rows)
```

Returns the intersection of two results

```
postgres=# select * from t_intersect1 INTERSECT select * from
t_intersect2;
```

Result:

```
 id | mc
----+-------
 1 | tbase
(1 row)
```

Returns the difference set of two results

```
postgres=# select * from t_except1 except select * from t_except2;
```

Result:

```
 id | mc
----+-------
 2 | tbase
(1 row)
```

Usage of any

Only needs to be greater than one of the values to be true:

```
postgres=# select * from t_any where id > any (select 1 union select 3);
```

Result:

```
 id | mc
----+-------
 2 | TBase
(1 row)
```

Usage of all

Must be greater than all values to be true:

```
postgres=# select * from t_all where id > all (select 1 union select 2);
```

Result:

```
 id | mc
----+-------
 3 | TBase
(1 row)
```

Aggregate Query

Count records

```
postgres=# select count(1) from tbase;
```

Result:

```
 count
-------
 3
(1 row)
```

Count distinct values in the record table

```
postgres=# select count(distinct id) from tbase;
```

Result:

```
 count
-------
 2
(1 row)
```

Sum

```
postgres=# select sum(id) from tbase;
```

Result:

```
 sum
-----
 4
(1 row)
```

Maximum value

```
postgres=# select max(id) from tbase;
```

Result:

```
 max
-----
 2
(1 row)
```

Minimum value

```
postgres=# select min(id) from tbase;
```

Result:

```
 min
-----
 1
(1 row)
```

Average value

```
postgres=# select avg(id) from tbase;
```

Result:

```
        avg
--------------------
 1.3333333333333333
(1 row)
```

Merge grouped fields into a single string

```
postgres=# create table t1(f1 int, f2 text, f3 text);
postgres=# insert into t1 values(1,'a','a');
postgres=# insert into t1 values(1,'b','b');
postgres=# insert into t1 values(2,'a','a');
postgres=# select f1, string_agg(f2, ',') from t1 group by f1;
```

Result:

```
 f1 | string_agg
----+------------
 1  | a,b
 2  | a
(2 rows)
```

Deduplication and custom aggregate function

Using `DISTINCT` and `ORDER BY` in aggregate functions requires meeting specific conditions. You can use custom functions to achieve more flexible aggregation.

Multi-table join

Inner join

```
postgres=# select * from tbase inner join t_appoint_col on
tbase.id=t_appoint_col.id;
```

Result:

```
 id | nickname | id | nickname
----+----------------------------+----+-------------
 1  | hello TBase                | 1  | hello TBase
 1  | The era of TBase distributed database has arrived | 1  | hello
TBase
(2 rows)
```

Left outer join

```
postgres=# select * from tbase left join t_appoint_col on
tbase.id=t_appoint_col.id;
```

Result:

```
 id | nickname    | id | nickname
----+-------------+----+-------------
 1  | hello TBase | 1  | hello TBase
 2  | TBase is good    |      |
 1  | The era of TBase distributed database has arrived | 1  | hello
TBase
(3 rows)
```

Right outer join

```
postgres=# select * from tbase right join t_appoint_col on
tbase.id=t_appoint_col.id;
```

Result:

```
 id | nickname                      | id | nickname
----+-------------------------------+----+-------------
 1  | The era of TBase distributed database has arrived | 1  | hello
TBase
 1  | hello TBase                   | 1  | hello TBase
    |                               5  | Power TBase
(3 rows)
```

Full join

```
postgres=# select * from tbase full join t_appoint_col on
tbase.id=t_appoint_col.id;
```

Result:

```
 id | nickname    | id | nickname
----+-------------+----+-------------
 1  | hello TBase | 1  | hello TBase
 2  | TBase is good    |      |
```

```
 1  | The era of TBase distributed database has arrived | 1  | hello
TBase
    |              | 5  | Power TBase
(4 rows)
```

Aggregation function concurrent computing

Single-core computing

```
postgres=# set max_parallel_workers_per_gather to 0;
postgres=# select count(1) from t_count;
```

Dual-core parallel

```
postgres=# set max_parallel_workers_per_gather to 2;
postgres=# select count(1) from t_count;
```

Quad-core parallel

```
postgres=# set max_parallel_workers_per_gather to 4;
postgres=# select count(1) from t_count;
```

not in clause includes null condition

```
postgres=# create table t_not_in(id int, mc text);
NOTICE: Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
postgres=# insert into t_not_in values(1, 'tbase'), (2, 'pgxz');
INSERT 0 2
postgres=# select * from t_not_in where id not in (3, 5);
```

Result:

```
 id | mc
----+-------
 1 | tbase
 2 | pgxz
(2 rows)
```

```
postgres=# select * from t_not_in where id not in (3, 5, null);
```

Result: No output row

Query data of specific data nodes (dn) only

```
postgres=# create table t_direct(id int, mc text);
NOTICE: Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
postgres=# insert into t_direct values(1, 'tbase'), (3, 'pgxz');
INSERT 0 2
postgres=# EXECUTE DIRECT ON (dn001) 'select * from t_direct';
```

Result:

```
 id | mc
----+-------
 1 | tbase
(1 row)
```

```
postgres=# EXECUTE DIRECT ON (dn002) 'select * from t_direct';
```

Result:

```
 id | mc
----+------
 3 | pgxz
(1 row)
```

Query data of all nodes:

```
postgres=# select * from t_direct;
```

Result:

```
 id | mc
```

```
----+-------
 1 | tbase
 3 | pgxz
(2 rows)
```

Special Application

Multiple rows to single row

```
postgres=# create table t_mulcol_tosimplecol(id int, mc text);
NOTICE: Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
postgres=# insert into t_mulcol_tosimplecol values(1, 'tbase'), (2,
'TBase');
INSERT 0 2
postgres=# select array_to_string(array(select mc from
t_mulcol_tosimplecol), ',');
```

Result:

```
 array_to_string
-----------------
 tbase,TBase
(1 row)
```

Single column to multiple rows

```
postgres=# create table t_col_to_mulrow(id int, mc text);
NOTICE: Replica identity is needed for shard table, please add to this
table through "alter table" command.
CREATE TABLE
postgres=# insert into t_col_to_mulrow values(1, 'tbase,TBase');
INSERT 0 1
postgres=# select regexp_split_to_table((select mc from t_col_to_mulrow
where id=1 limit 1), ',');
```

Result:

```
 regexp_split_to_table
-----------------------
```

```
 tbase
 TBase
(2 rows)
```

Query the data node of the record

```
postgres=# select xc_node_id, * from t1;
```

Result:

```
 xc_node_id | f1 | f2
------------+----+----
 2142761564 | 1  | 3
 2142761564 | 1  | 3
(2 rows)
```

Query and map node name:

```
postgres=# select t1.xc_node_id, pgxc_node.node_name, t1.* from t1,
pgxc_node where t1.xc_node_id=pgxc_node.node_id;
```

Result:

```
 xc_node_id | node_name | f1 | f2
------------+-----------+----+----
 2142761564 | dn001     | 1  | 3
 2142761564 | dn001     | 1  | 3
(2 rows)
```

Grouping Sets/Rollup/Cube Usage

GROUP BY Usage

Create Sales Detail Table and Insert Data:

```
create table t_grouping(id int, dep varchar(20), product varchar(20),
num int);
INSERT INTO t_grouping VALUES(1, 'Business Unit 1', 'Mobile Phone', 90);
-- More data inserts ...
```

Group Summary by Department and Product:

```
postgres=# select dep, product, sum(num) from t_grouping group by dep,
product order by dep, product;
```

Result:

```
 dep          | product     | sum
--------------+-------------+-----
 Business Unit 1 | Computer        | 80
 Business Unit 1 | Mobile Phone    | 160
 Business Unit 2 | Computer        | 120
 Business Unit 2 | Mobile Phone    | 50
 Business Unit 3 | Computer        | 80
 Business Unit 3 | Mobile Phone    | 160
```

Group using Grouping Sets:

```
postgres=# select dep, product, sum(num) from t_grouping group by
grouping sets((dep), (product), ());
```

Result:

```
 dep          | product     | sum
--------------+-------------+-----
 Business Unit 1 | (null)          | 240
 Business Unit 2 | (null)          | 170
 Business Unit 3 | (null)          | 240
 Computer        | (null)          | 280
 Mobile Phone    | (null)          | 370
(null)         | (null)          | 650
```

Using rollup and cube:

```
postgres=# select dep, product, sum(num) from t_grouping group by
rollup((dep), (product));
postgres=# select dep, product, sum(num) from t_grouping group by
cube((dep), (product));
```

Results are the same as grouping sets.

PREPARE preparation usage

Create a prepared statement

```
postgres=# create table t1(f1 int, f2 int);
CREATE TABLE
postgres=# insert into t1 values(1, 1), (2, 2);
COPY 2
postgres=# PREPARE usrrptplan (int) AS SELECT * FROM t1 WHERE f1=$1;
PREPARE
postgres=# EXECUTE usrrptplan(1);
```

Result:

```
 f1 | f2
----+----
 1  | 1
(1 row)
```

Deallocate a prepared statement

```
postgres=# DEALLOCATE usrrptplan;
DEALLOCATE
postgres=# EXECUTE usrrptplan(1);
ERROR: prepared statement "usrrptplan" does not exist
```

INSERT statement

Last updated: 2024-08-22 17:00:40

SQL INSERT statement

Insert a single record

- Specify all fields:

```
postgres=# insert into tbase(id, nickname) values(1, 'hello TBase');
```

Result: `INSERT 0 1`

- Specify some fields; unspecified fields will use default values:

```
postgres=# insert into tbase(nickname) values('TBase 好');
```

Result: `INSERT 0 1`

- If no fields are specified, all fields will be as defined during table creation:

```
postgres=# insert into tbase values(nextval('t_id_seq'::regclass),
'TBase 好');
```

Result: `INSERT 0 1`

- The order of fields can be arranged arbitrarily:

```
postgres=# insert into tbase(nickname, id) values('TBase swap', 5);
```

Result: `INSERT 0 1`

- Use the `default` keyword to set values to their default as specified during table creation:

```
postgres=# insert into tbase(id, nickname) values(default, 'TBase
default');
```

Result: `INSERT 0 1`

- Data after inserting the records:

```
postgres=# select * from tbase;
```

id	nickname
1	hello TBase
2	TBase OK
5	TBase swap
3	TBase OK
4	TBase default

Insert multiple records

- Insert multiple records:

```
postgres=# insert into tbase(id, nickname) values(1, 'hello TBase'),
(2, 'TBase OK');
```

Result: `INSERT 0 2`

- Query result after insertion:

```
postgres=# select * from tbase;
```

id	nickname
1	hello TBase
2	TBase OK

Insert data using subquery

- Insert data using subquery:

```
postgres=# insert into tbase(id, nickname) values(1, (select relname
from pg_class limit 1));
```

Result: `INSERT 0 1`

- Query result after insertion:

```
postgres=# select * from tbase;
```

id	nickname
1	pg_statistic

Batch insert data from another table

- Batch insert data from another table:

```
postgres=# insert into tbase(nickname) select relname from pg_class
limit 3;
```

Result: `INSERT 0 3`

- Query result after insertion:

```
postgres=# select * from tbase;
```

id	nickname
5	pg_type
6	pg_toast_2619
4	pg_statistic

Generating data in bulk

- Using the `generate_series` function to generate data in bulk:

```
postgres=# insert into tbase select t, md5(random()::text) from
generate_series(1,10000) as t;
```

Result: `INSERT 0 10000`

- Count after insertion:

```
postgres=# select count(1) from tbase;
```

count
10000

Return the inserted data, easily get the inserted record's serial value

- Insert record and return all fields:

```
postgres=# insert into tbase(nickname) values('TBase OK') returning *;
```

id	nickname
7	TBase OK

- Specify return fields:

```
postgres=# insert into tbase(nickname) values('hello TBase') returning
id;
```

id
8

Insert..Update update

- Use the `ON CONFLICT` clause to update:

```
postgres=# insert into t values(1,'pgxz') ON CONFLICT (id) DO UPDATE
SET nc = 'tbase';
```

Result: `INSERT 0 1`

- Query result after update:

```
postgres=# select * from t;
```

id	nc
1	tbase

insert all

- Use the `insert all` statement:

```
postgres=# create table t5(f1 int, f2 int);
postgres=# create table t6(f1 int, f2 int);
postgres=# insert all into t5 values(1, 1) into t6 values(1, 1) select
1 as f1, 1 as f2;
```

Result: `INSERT 0 2`

- **Query t5 table:**

```
postgres=# select * from t5;
```

f1	f2
1	1

- **Query t6 table:**

```
postgres=# select * from t6;
```

f1	f2
1	1

update statement

Last updated: 2024-08-22 17:01:27

Single table update

```sql
update tbase set nickname ='Hello TBase' where id=1;
```

Condition expression method

```sql
update tbase set nickname = 'Good TBase' where nickname is null;
```

Query result:

```sql
select * from tbase;
```

id	nickname
2	TBase is good
1	Hello TBase
3	Good TBase

Distributed keys cannot be updated:

```sql
update t2 set f1=1 where f2=1;
```

Error message:

```
ERROR: Distributed column "f1" can't be updated in current version
```

Multi-table join update

```sql
update tbase set nickname ='Good TBase' from t_appoint_col where
t_appoint_col.id=tbase.id;
```

Query result:

```
select * from tbase;
```

id	nickname
2	TBase is good

Return updated data

```
update tbase set nickname = nickname where id = (random()*2)::integer
returning id, nickname;
```

Returned result:

id	nickname
2	TBase is good

Multi-column matching update

```
update tbase set (nickname, age) = ('TBase is good',
(random()*2)::integer);
```

Query result:

```
select * from tbase;
```

id	nickname	age
1	TBase is good	2
2	TBase is good	0

Shard Key is not allowed to be updated

```
update tbase set id=8 where id=1;
```

Error message:

```
ERROR: Distribute column or partition column can't be updated in current
version
```

DELETE Statement

Last updated: 2024-08-22 17:02:21

Conditional Deletion

```sql
select * from tbase;
```

id	nickname
2	TBase is good
1	Hello TBase
3	
4	TBase good

```sql
delete from tbase where id=4;
```

Null Condition Expression

```sql
delete from tbase where nickname is null;
```

```sql
select * from tbase;
```

id	nickname
2	TBase is good
1	Hello TBase

Multi-table Join Data Deletion

```sql
set prefer_olap to on;
```

```sql
delete from tbase using t_appoint_col where tbase.id=t_appoint_col.id;
```

```
select * from tbase;
```

id	nickname
2	TBase is good

Return Deleted Data

```
delete from tbase returning *;
```

id	nickname
2	TBase is good

Delete All Data

```
insert into tbase select t,random()::text from generate_series(1,100000)
as t;
```

```
truncate table tbase;
```

Usage of cursors

Last updated: 2024-08-22 17:02:50

Definition of a cursor

```
begin;
```

```
DECLARE tbase_cur SCROLL CURSOR FOR SELECT * from tbase ORDER BY id;
```

> **Note:**
> Cursors need to be used within a transaction

Extract the next row of data

```
FETCH NEXT from tbase_cur ;
```

id	nickname
1	hello TBase

```
FETCH NEXT from tbase_cur ;
```

id	nickname
2	TBase is good

Extract the previous row of data

```
FETCH PRIOR from tbase_cur ;
```

id	nickname
1	hello TBase

```
FETCH PRIOR from tbase_cur ;
```

Extract the last row

```
FETCH LAST from tbase_cur ;
```

id	nickname
5	TBase swap

Extract the first row

```
FETCH FIRST from tbase_cur ;
```

id	nickname
1	hello TBase

Extract the x-th row of the query

```
FETCH ABSOLUTE 2 from tbase_cur ;
```

id	nickname
2	TBase is good

```
FETCH ABSOLUTE -1 from tbase_cur ;
```

id	nickname
5	TBase swap

Extract the next x rows from the current position

```
FETCH ABSOLUTE 1 from tbase_cur ;
```

id	nickname
1	hello TBase

```
FETCH RELATIVE 2 from tbase_cur ;
```

id	nickname
3	TBase is good

Extract x rows forward

```
FETCH FORWARD 2 from tbase_cur ;
```

id	nickname
1	hello TBase
2	TBase is good

```
FETCH FORWARD 2 from tbase_cur ;
```

id	nickname
3	TBase is good
4	TBase default

Extract all remaining rows forward

```
FETCH FORWARD all from tbase_cur ;
```

id	nickname
3	TBase is good
4	TBase default
5	TBase swap

Extract x rows backward

```
FETCH BACKWARD 2 from tbase_cur ;
```

id	nickname
5	TBase swap
4	TBase default

Extract all remaining rows backward

```
FETCH BACKWARD all from tbase_cur ;
```

id	nickname
3	TBase is good
2	TBase is good
1	hello TBase

Usage of COPY

Last updated: 2024-08-22 17:03:13

COPY is used for mutual data copying between TBase tables and standard file system files. COPY TO can copy the contents of a table to a file, and COPY FROM can copy data from a file to a table (data is appended to the existing data). COPY TO can also copy the results of a SELECT query to a file. If a column list is specified, COPY will only copy the data of the specified columns to or from the file. If there are columns in the table not listed, COPY FROM will insert default values for those columns. When using COPY, the TBase server reads from or writes to a file directly from "local" storage. The file must be accessible to the TBase user (the user ID running the server) and should be named from the server's perspective.

Experiment Table Structure and Data

```sql
\d+ t
```

Column	Type	Modifiers	Storage	Stats target	Description
f1	integer	not null	plain		
f2	character	not null	extended		
f3	timestamp	default now()	plain		
f4	integer		plain		

The data testing process can re-enter and modify rows

```sql
select * from t;
```

f1	f2	f3	f4
3	pgxz	2017-10-28 18:24:05.645691	
1	Tbase		7
2		2017-10-28 18:24:05.643102	3

COPY TO Detailed Usage – Copy Data to a File

Export all columns

```
copy public.t to '/data/pgxz/t.txt';
```

```
cat /data/pgxz/t.txt
```

The default generated file content includes all columns of the table, with columns separated by tabs. NULL values are represented as \N.

Export selected columns

```
copy public.t(f1,f2) to '/data/pgxz/t.txt';
```

```
cat /data/pgxz/t.txt
```

Only export the f1 and f2 columns

Export query results

```
copy (select f2,f3 from public.t order by f3) to '/data/pgxz/t.txt';
```

```
cat /data/pgxz/t.txt
```

The query can be any complex query

Specify file format

Generate in CSV format

```
```sql
copy public.t to '/data/pgxz/t.txt' with csv;
```

```
cat /data/pgxz/t.txt
```

Generate in binary format
```

```
copy public.t to '/data/pgxz/t.txt' with binary;
```

Default format is TEXT

Specify the delimiter between columns using the delimiter option

```
copy public.t to '/data/pgxz/t.txt' with delimiter '@';
```

```
cat /data/pgxz/t.txt
```

Specify the character that separates columns in the file. The default is a tab character for text format and a comma for CSV format. The delimiter must be a single-byte character, meaning Chinese characters are not supported. This option is not allowed when using binary format.

Handling NULL values

```
copy public.t to '/data/pgxz/t.txt' with NULL 'NULL';
```

```
cat /data/pgxz/t.txt
```

When a value is NULL, it is exported as the NULL string. This option is not allowed when using binary format.

Generate column header names

```
Export the table using CSV format and the HEADER option:
```sql
copy public.t to '/data/pgxz/t.txt' with csv HEADER;
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
f1,f2,f3,f4
1,Tbase,,7
2,pgxc,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```
```

Attempting to use the HEADER option in non-CSV mode will result in an error:

```
copy public.t to '/data/pgxz/t.txt' with HEADER;
```

```
ERROR: COPY HEADER available only in CSV mode
```

Export the oids system column

Create a new table with oids:

```
drop table t;
CREATE TABLE t (
    f1 integer NOT NULL,
    f2 text NOT NULL,
    f3 timestamp without time zone,
    f4 integer
) with oids DISTRIBUTE BY SHARD (f1);
```

Import data including oids:

```
copy t from '/data/pgxz/t.txt' with csv;
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg'', xc	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

Export data using the oids option:

```
copy t to '/data/pgxz/t.txt' with oids;
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
35055 1 Tbase \N 7
35056 2 pg'", xc 2017-10-28 18:24:05.643102 3
35177 3 pgxz 2017-10-28 18:24:05.645691 \N
```

Use the quote option to quote characters from the definition

Default CSV export reference characters are double quotes:

```
copy t to '/data/pgxz/t.txt' with csv;
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
1,Tbase,,7
2,"pg'"", xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

Custom reference characters are percentage signs:

```
copy t to '/data/pgxz/t.txt' with quote '%' csv;
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
1,Tbase,,7
2,%pg'", xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

Use custom escape characters:

If escape is not specified, it defaults to the QUOTE value:

```
copy t to '/data/pgxz/t.txt' with quote '%' csv;
```

Custom escape character is '@':

```
copy t to '/data/pgxz/t.txt' with quote '%' escape '@' csv;
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
1,Tbase,,7
2,%pg'", xc@%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

Attempting to use multibyte characters as escape will result in an error:

```
copy t to '/data/pgxz/t.txt' with quote '%' escape '@@' csv;
```

```
ERROR: COPY escape must be a single one-byte character
```

Force add reference characters to a specific column

```
Export specified columns using CSV format and force_quote option:
```sql
copy t to '/data/pgxz/t.txt' (format 'csv', force_quote (f1,f2));
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
"1","Tbase",,7
"2","pg'"", xc%",2017-10-28 18:24:05.643102,3
"3","pgxz",2017-10-28 18:24:05.645691,
```

Specify multiple columns to force the addition of reference characters; column order can be arbitrary:
```

```
copy t to '/data/pgxz/t.txt' (format 'csv', force_quote (f1,f4,f3,f2));
```

View the export results:

```
cat /data/pgxz/t.txt
```

```
"1","Tbase",,"7"
"2","pg'"", xc%","2017-10-28 18:24:05.643102","3"
"3","pgxz","2017-10-28 18:24:05.645691",
```

Attempting to use the `force_quote` option in non-CSV mode will result in an error:

```
copy t to '/data/pgxz/t.txt' (format 'text', force_quote (f1,f2,f3,f4));
```

```
ERROR: COPY force quote available only in CSV mode
```

Use encoding to specify file content encoding for export

Export files using different encodings:

```
copy t to '/data/pgxz/t.csv' (encoding utf8);
copy t to '/data/pgxz/t.csv' (encoding gbk);
```

Set client encoding with `set_client_encoding` and export CSV files:

```
set client_encoding to gbk;
copy t to '/data/pgxz/t.csv' with csv;
```

Use multibyte characters as separators

Tencent Cloud Data Warehouse TCHouse-P version 5.06.2.1 and above supports multibyte separators.
Create a new table and insert data:

```
create table t_position_copy(f1 int, f2 varchar(10), f3 int);
insert into t_position_copy values(123, 'Adi', 2021);
```

Export data using multibyte separators:

```
copy t_position_copy to '/data/tbase/t_position_copy_mul_delimiter.text'
with delimiter '@\u0026';
```

View the export results:

```
cat /data/tbase/t_position_copy_mul_delimiter.text
```

```
123@\u0026Adi@\u00262021
```

COPY FROM Detailed Usage Explanation—Copy file content to the data table

Import all columns

```
cat /data/pgxz/t.txt
```

```
truncate table t;
copy t from '/data/pgxz/t.txt';
```

```
select * from t;
```

Import specified columns

```
Copy data from specified columns to a file:
```sql
copy t(f1,f2) to '/data/pgxz/t.txt';
```

View file content:

```
cat /data/pgxz/t.txt
```

```
1 Tbase
2 pg'", xc%
```
```

```
3 pgxz
```

Clear table `t` and import specified columns from a file:

```
truncate table t;
copy t(f1,f2) from '/data/pgxz/t.txt';
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase	2017-10-30 11:54:16.559579	
2	pg'	2017-10-30 11:54:16.559579	
3	pgxz	2017-10-30 11:54:16.560283	

Specify the input file format

Import text file:

```
copy t from '/data/pgxz/t.txt' (format 'text');
```

Import CSV file:

```
copy t from '/data/pgxz/t.csv' (format 'csv');
```

Import binary file:

```
copy t from '/data/pgxz/t.bin' (format 'binary');
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4

Tbase			7
pg'", xc%	2017-10-28 18:24:05.643102	3	
pgxz	2017-10-28 18:24:05.645691		

Specify the delimiter between columns using the delimiter option

Import text file using a custom delimiter:

```
copy t from '/data/pgxz/t.txt' (format 'text', delimiter '%');
```

Import CSV file using custom delimiter:

```
copy t from '/data/pgxz/t.csv' (format 'csv', delimiter '%');
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg'", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

NULL value handling

The import file contains NULL strings:

```
copy t from '/data/pgxz/t.txt' (null 'NULL');
```

Query to confirm NULL value handling results:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase		7

2	pg'", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

Note:
The string NULL may not be recognized in different contexts, leading to import errors.

```
copy t from '/data/pgxz/t.txt' (null 'null');
```

```
ERROR: invalid input syntax for type timestamp: "NULL"
CONTEXT: COPY t, line 1, column f3: "NULL"
```

Custom quote character

```
View CSV file content:
```shell
cat /data/pgxz/t.csv
```

```
1,Tbase,,7
2,%pg'", xc%%%,2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

Configure quote character when importing CSV file:

```
copy t from '/data/pgxz/t.csv' (format 'csv', quote '%');
```

Custom escape character

View CSV file content:

```
cat /data/pgxz/t.csv
```

```
1,Tbase,,7
2,"pg'@", xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```
```

Configure escape character when importing CSV file:

```
copy t from '/data/pgxz/t.csv' (format 'csv', escape '@');
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg'", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	

CSV Header ignore first line

View CSV file content (with header):

```
cat /data/pgxz/t.csv
```

```
f1,f2,f3,f4
1,Tbase,,7
2,"pg'"", xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

Ignore header when importing CSV file:

```
copy t from '/data/pgxz/t.csv' (format 'csv', header true);
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4
Tbase			7
pg'", xc%	2017-10-28 18:24:05.643102	3	

pgxz	2017-10-28 18:24:05.645691		

Import oid column value

View text file content (including oid):

```
cat /data/pgxz/t.txt
```

```
35242 1 Tbase \N 7
35243 2 pg'", xc% 2017-10-28 18:24:05.643102 3
35340 3 pgxz 2017-10-28 18:24:05.645691 \N
```

Import text file and retain oid:

```
truncate table t;
copy t from '/data/pgxz/t.txt' (oids true);
```

Query to confirm the import results:

```
select oid, * from t;
```

oid	f1	f2	f3	f4
35242	1	Tbase		7
35243	2	pg'", xc%	2017-10-28 18:24:05.643102	3
35340	3	pgxz	2017-10-28 18:24:05.645691	

Use FORCE_NOT_NULL to convert null values in a column to zero-length strings

View CSV file content (including null values):

```
cat /data/pgxz/t.csv
```

```
1,Tbase,,7
2,"pg'""", xc%",2017-10-28 18:24:05.643102,3
3,pgxz,2017-10-28 18:24:05.645691,
```

```
4,,2017-10-30 16:14:14.954213,4
```

Import CSV file using FORCE_NOT_NULL:

```
truncate table t;
copy t from '/data/pgxz/t.csv' (format 'csv', FORCE_NOT_NULL (f2));
```

Query to confirm the null value handling result in a column:

```
select * from t where f2 = '';
```

f1	f2	f3	f4
4		2017-10-30 16:14:14.954213	4

Specify import file encoding with Encoding

```
Determine file encoding using the enca command:
```shell
enca -L zh_CN /data/pgxz/t.txt
```

```
Simplified Chinese National Standard; GB2312
```

Import file into the database:

```
copy t from '/data/pgxz/t.txt';
```

Query to confirm the import results:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg'", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	
```

If the encoding format of the import file is not specified, Chinese characters cannot be imported correctly. Reimport using GBK encoding:

```
truncate table t;
copy t from '/data/pgxz/t.txt' (encoding gbk);
```

Query to confirm import results again:

```
select * from t;
```

f1	f2	f3	f4
1	Tbase		7
2	pg'", xc%	2017-10-28 18:24:05.643102	3
3	pgxz	2017-10-28 18:24:05.645691	
4	Tencent	2017-10-30 16:41:09.157612	4

You can use the `enconv` command to convert the file encoding before importing data:

```
truncate table t;
enconv -L zh_CN -x UTF-8 /data/pgxz/t.txt
copy t from '/data/pgxz/t.txt';
```

Query to confirm the import results after converting encoding:

```
select * from t;
```

Use Position to specify field length for import

View the content of the text file to be imported:

```
cat /data/tbase/t_position_copy.txt
```

```
123 Adi 2021
```

Use position to specify field length for import:

```
copy t_position_copy (f1 position(1:3), f2 position(4:9), f3
position(10:13)) from '/data/tbase/t_position_copy.txt';
```

Position specifies field length and function for import

```
```shell
cat /data/tbase/t_position_copy.txt
```

```
123 Adi 2020010120200101121212
```

```
copy t_position_copy (f1 position(1:3), f2 position(4:9), f3
position(10:17) to_date($3,'yyyymmdd'), f4 position(18:31)
to_timestamp($4,'yyyymmddhh24miss')) from
'/data/tbase/t_position_copy.txt';
```

Position specifies the encoding to be used

Tencent Cloud Data Warehouse TCHouse-P version 5.06.2.1 and above supports this feature. This feature allows data files with character encoding different from the database to be imported correctly.

```
file /data/tbase/t_position_copy.txt
```

```
/data/tbase/t_position_copy.txt: ISO-8859 text
```

Importing data without specifying encoding will cause errors:

```
copy t_position_copy (f1 position(1:3), f2 position(4:7), f3
position(8:11)) from '/data/tbase/t_position_copy.txt';
```

```
ERROR: multibyte character "Ü2" is truncated for column f2
CONTEXT: COPY t_position_copy, line 1: "123°¢µÜ2021",
nodetype:1(1:cn,0:dn)
```

Specify the file encoding to be imported as gbk:
```

```
copy t_position_copy (f1 position(1:3), f2 position(4:7), f3
position(8:11)) from '/data/tbase/t_position_copy.txt' encoding 'gbk';
```

```
select * from t_position_copy;
```

f1	f2	f3
123	Adi	2021

Note, in gbk encoding, the length of one Chinese character is 2.
If your gbk file is exported from Oracle, then import it into the utf database of TBase to ensure correct import:

```
copy t4(f1 position(1:1), f2 position(2:3)) from '/data/tbase/t4.txt'
with csv encoding 'gbk';
```

If not specified, importing will encounter errors with characters such as " ", " ":

```
copy t4(f1 position(1:1), f2 position(2:3)) from '/data/tbase/t4.txt'
encoding 'gbk';
```

```
ERROR: character with byte sequence 0xab 0x0a in encoding "GBK" has no
equivalent in encoding "UTF8"
CONTEXT: COPY t4, line 1: "1«1ÌÚ", nodetype:1(1:cn,0:dn)
```

Use multibyte separators

Tencent Cloud Data Warehouse TCHouse-P version 5.06.2.1 and above is supported.

```
create table t_position_copy(f1 int, f2 varchar(10), f3 int);
```

```
cat /data/tbase/t_position_copy_mul_delimiter.text
```

```
123@\u0026Adi@\u00262021
```

```
copy t_position_copy from
'/data/tbase/t_position_copy_mul_delimiter.text' with delimiter
'@\u0026';
```

```
select * from t_position_copy;
```

f1	f2	f3
123	Adi	2021

Usage of JSON/JSONB

Last updated: 2024-08-22 17:03:34

TBase is not just a distributed relational database system; it also supports non-relational data types like JSON. The JSON data type is used to store JSON (JavaScript Object Notation) data. This data can also be stored as text, but the advantage of the JSON data type is that it mandates each stored value to conform to JSON rules. There are many JSON-related functions and operators that can be used to store data in these data types. The JSON data type has two variants: json and jsonb. They accept the exact same set of values as input. The main practical difference between the two is efficiency.

JSON Applications

Creating tables with JSON type fields

```sql
create table t_json(id int,f_json json);
```

Inserting Data

```sql
insert into t_json values(1,'{"col1":1,"col2":"tbase"}');
insert into t_json values(2,'{"col1":1,"col2":"tbase","col3":"pgxz"}');
```

Obtaining JSON object domains through keys

```sql
select f_json ->'col2' as col2 ,f_json -> 'col3' as col3 from t_json;
```

col2	col3
"tbase"	
"tbase"	"pgxz"

Retrieving object values in text form

```sql
select f_json ->>'col2' as col2 ,f_json ->> 'col3' as col3 from t_json;
```

col2	col3
tbase	
tbase	pgxz

JSONB Applications

Creating tables with JSONB type fields

```
create table t_jsonb(id int,f_jsonb jsonb);
```

Inserting Data

```
insert into t_jsonb values(1,'{"col1":1,"col2":"tbase"}');
insert into t_jsonb values(2,'{"col1":1,"col2":"tbase","col3":"pgxz"}');
```

- JSONB removes duplicate keys upon insertion.

Updating Data

- Adding elements

```
update t_jsonb set f_jsonb = f_jsonb || '{"col3":"pgxz"}'::jsonb where
id=1;
```

- Updating existing elements

```
update t_jsonb set f_jsonb = f_jsonb || '{"col2":"tbase"}'::jsonb where
id=3;
```

- Deleting a key

```
update t_jsonb set f_jsonb = f_jsonb - 'col3';
```

Updating data with the jsonb_set() function

```
update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col}', '"pgxz"' ,
true ) where id=1;
update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col}', '"pgxz"' ,
false ) where id=2;
update t_jsonb set f_jsonb = jsonb_set( f_jsonb , '{col2}', '"pgxz"' ,
false ) where id=3;
```

JSONB function applications

jsonb_each() converts JSON objects into keys and values

```
select * from jsonb_each((select f_jsonb from t_jsonb where id=1));
```

key	value
col	"pgxz"
col1	1
col2	"tbase"

jsonb_each_text() converts JSON objects into text type keys and values

```
select * from jsonb_each_text((select f_jsonb from t_jsonb where id=1));
```

key	value
col	pgxz
col1	1
col2	tbase

row_to_json() converts a row record into a JSON object

```
select row_to_json(tbase) from tbase;
```

```
{"id":1,"nickname":"tbase"}
```

```
{"id":2,"nickname":"pgxz"}
```

json_object_keys() returns all keys in an object

```
select * from json_object_keys((select f_jsonb from t_jsonb where
id=1)::json);
```

json_object_keys
col
col1
col2

JSONB index usage

TBase provides GIN indexes for JSONB documents, which can be used to efficiently search for keys or key-value pairs in large JSONB documents (data).

Create JSONB index

```
create index t_jsonb_f_jsonb_idx on t_jsonb using gin(f_jsonb);
```

Performance testing the query

- No index overhead

```
select * from t_jsonb where f_jsonb @> '{"col1":9999}';
```

- With index overhead

```
select * from t_jsonb where f_jsonb @> '{"col1":9999}';
```

with the use of Subquery and Recursion

Last updated: 2024-08-22 17:03:52

with the use of Subquery and Recursion

```sql
create table t_area(id int,pid int,area varchar);
insert into t_area values(1,null,'Guangdong Province');
insert into t_area values(2,1,'Shenzhen City');
insert into t_area values(3,1,'Dongguan City');
insert into t_area values(4,2,'Nanshan District');
insert into t_area values(5,2,'Futian District');
insert into t_area values(6,2,'Luohu District');
insert into t_area values(7,3,'Nancheng District');
insert into t_area values(8,3,'Dongcheng District');
```

with Subquery

Using `with` Subquery to select all rows with `pid=1` :

```sql
with t_gd_city AS (
  select * from t_area where pid=1
)
select * FROM t_gd_city;
```

Query result:

id	pid	area
2	1	Shenzhen City
3	1	Dongguan City