

# Alibaba Cloud HybridDB for PostgreSQL

## User Guide

Issue: 20190221

## Legal disclaimer

---

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

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use








or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).
6. Please contact Alibaba Cloud directly if you discover any errors in this document.



## Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 <b>Danger:</b> Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 <b>Warning:</b> Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 <b>Notice:</b> Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 <b>Note:</b> You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
<b>Bold</b>	It is used for buttons, menus, page names, and other UI elements.	Click OK.
Courier font	It is used for commands.	Run the <code>cd /d C:/windows</code> command to enter the Windows system folder.
<i>Italics</i>	It is used for parameters and variables.	<code>bae log list --instanceid Instance_ID</code>
[ ] or [a b]	It indicates that it is an optional value, and only one item can be selected.	<code>ipconfig [-all -t]</code>

Style	Description	Example
<code>{}</code> or <code>{a b}</code>	It indicates that it is a required value, and only one item can be selected.	<code>switch {stand   slave}</code>



# Contents

---

Legal disclaimer.....	I
Generic conventions.....	I
1 Basic operations.....	1
2 Manage instances.....	4
2.1 Create an instance.....	4
2.2 Apply for an Internet address.....	4
2.3 Release an Internet IP address.....	6
2.4 Upgrade the instance configuration.....	7
2.5 Change connection addresses.....	9
2.6 Restart an instance.....	9
2.7 Release an instance.....	10
3 Migrate data.....	12
4 User and permission management.....	13
5 Extension management.....	15
6 Operations of JSON data.....	17
7 Use HyperLogLog.....	25
8 Use the create Library command.....	27
9 Use PL / Java UDF.....	29
10 Use SortKey.....	31



# 1 Basic operations

---

HybridDB for PostgreSQL is consistent with Greenplum Database in operations based on Greenplum Database, including the schema, supported types, and user permissions. Aside from some operations exclusive to Greenplum Database, such as the Distribution Key and AO table, you can refer to PostgreSQL for all the other operations.

This document introduces the basic operations of HybridDB for PostgreSQL, including [creating a database](#), [creating a distribution key](#), [constructing data](#), and [creating a query](#).

## Create a database

In HybridDB for PostgreSQL instances, you can use SQL statements to create a database following the same operations in PostgreSQL. For example, after an instance is connected to Greenplum through the psql tool, run the following command to create a database:

```
=> create database mygpdb;  
CREATE DATABASE  
=> \c mygpdb  
psql (9.4.4, server 8.3devel)  
You are now connected to database "mygpdb" as user "mygpdb".
```

## Create a distribution key

In HybridDB for PostgreSQL, tables are distributed on all of the Segments following a hash or random distribution rule. You can specify the distribution key when creating a table. By doing this, data imported is assigned to the specific Segment according to the hash value calculated by the distribution key.

```
=> create table vtbl(id serial, key integer, value text, shape cuboid  
, location geometry, comment text) distributed by (key);  
CREATE TABLE
```

When no distribution key is specified, that is, with no “distributed by (key)” followed in the command, Greenplum randomizes the ID field using the round-robin approach.

## Best practices

Distribution keys are vital to query performance. When you are specifying distribution on keys, we recommend that you follow the “Even” principle. What’s more, specifying a more business-cued field can significantly improve the performance.

To be specific, the best practices include:

- Select the evenly distributed columns or multiple columns to prevent data from tilting.
- Select the fields commonly used in JOIN, especially for statements with high concurrency.
- Select the condition columns with high concurrent query and high filter rate.
- Do not use random distribution.

For details, see [Reference](#).

## Construct data

### 1. Create a function to generate a random string.

```
CREATE OR REPLACE FUNCTION random_string(integer) RETURNS text AS $
body$
SELECT array_to_string(array
                        (SELECT substring('0123456789ABCDEFGHIJ
KLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
FROM (ceil(random()*62))::
int
FOR 1)
FROM generate_series(1, $1)), '');
$body$
LANGUAGE SQL VOLATILE;
```

### 2. Create a distribution key.

```
CREATE TABLE tbl(id serial, KEY integer, locate geometry, COMMENT
text) distributed by (key);
```

### 3. Construct data.

```
INSERT INTO tbl(KEY, COMMENT, locate)
SELECT
    KEY,
    COMMENT,
    ST_GeomFromText(locate) AS locate
FROM
    (SELECT
        (a + 1) AS KEY,
        random_string(ceil(random() * 24)::integer) AS COMMENT,
        'POINT(' || ceil(random() * 36 + 99) || ' ' || ceil(random
() * 24 + 50) || ' ' AS locate
    FROM
        generate_series(0, 99999) AS a)
    AS t;
```

## Create a query

- Query

```
=> select * from tbl where key = 751;
```

```

| id | key | value | shape | locate
+----+----+-----+-----+-----+
| 751 | 751 | red   | 010100000000000000000000C05B400000000000004A40
| B9hPhjeNWPqV |
(1 row)
Time: 513.101 ms

```

- Query plan

```

=> explain select * from tbl where key = 751;
 Gather Motion 1:1  (slice1; segments: 1)  (cost=0.00..1519.28 rows
=1 width=53)
   -> Seq Scan on tbl  (cost=0.00..1519.28 rows=1 width=53)
       Filter: key = 751
Settings:  effective_cache_size=8GB; gp_statistics_use_fkeys=on
Optimizer status: legacy query optimizer

```

## Reference

- [Pivotal Greenplum Official Documentation](#)
- [Greenplum 4.3 Best Practices](#)

## 2 Manage instances

---

### 2.1 Create an instance

For the more information, see [Create an instance](#).

### 2.2 Apply for an Internet address

If your application is deployed on an ECS instance in the same region as your HybridDB for PostgreSQL instance and has the same [network type](#), you do not need to apply for an Internet address. If your application is deployed on an ECS that is located in a different region or has a network type different from your HybridDB for PostgreSQL instance, or is deployed on a platform other than Alibaba Cloud, an Internet address is necessary for access to the HybridDB for PostgreSQL instance.



**Note:**

Instances in the same region (can be in different zones) with the same network type can access each other through the intranet network.

#### Scenarios

Internet addresses and intranet addresses are used in the following scenarios:

- Use an intranet addresses only:
  - Your application is deployed on an ECS instance in the same region as your HybridDB for PostgreSQL instance and shares the same [network type](#) with the ECS instance.
- Use an Internet addresses only:
  - The ECS instance where your application is deployed and your HybridDB for PostgreSQL instance are in different regions.
  - Your application is deployed in a third-party system other than Alibaba Cloud.

- Use both Internet and intranet IP addresses:
  - Some application modules are deployed on an ECS instance in the same region with the same *network type*, while other modules are deployed on an ECS instance in a different region.
  - Some modules of the application are deployed on an ECS instance in the same region with the same *network type*, while other modules are deployed in systems other than Alibaba Cloud.

## Notes

- Before connecting to the database, you must add the IP addresses or IP ranges to a whitelist. For more information, see [Set up a whitelist](#).
- Exercise caution when you select an Internet address, because the instance may be exposed to security risks. To reach a higher transmission rate and higher security level, we recommend you transfer your applications to the ECS instance located in the same region as the HybridDB for PostgreSQL instance.

## Procedure

1. Log on to the [HybridDB for PostgreSQL console](#).
2. Select the Region of the instance.
3. Locate the target instance. In the Actions column, click Manage.
4. On the Basic Information page, click Apply for internet address to go to the Database Connection page. You can also directly click Database Connection on the left-side pane.
5. On the Database Connection page, click Apply for internet address.
6. On the dialog box that appears, click OK to generate an Internet address.

After allocating the Internet address, you can click Release Internet Address on the Database Connection page to release the Internet address.

## Related API

API	Description
<a href="#">#unique_14</a>	Apply for an Internet address.

## 2.3 Release an Internet IP address

If the network environment changed after the Internet address is allocated, you can release the Internet address on HybridDB for PostgreSQL console if you don't need it any more. After releasing the Internet address, make sure to change the application configurations which related to this address.

Before performing this operation, please read the following scenarios.

### Scenarios

Internet IP addresses and intranet IP addresses are used in the following scenarios:

- Use an intranet IP addresses only:
  - Your application is deployed on an ECS instance in the same region as your HybridDB for PostgreSQL instance and shares the same *network type* with the ECS instance.
- Use an Internet IP addresses only:
  - The ECS instance where your application is deployed and your HybridDB for PostgreSQL instance are in different regions.
  - Your application is deployed in a third-party system other than Alibaba Cloud.
- Use both Internet and intranet IP addresses:
  - Some application modules are deployed on an ECS instance in the same region with the same *network type*, while other modules are deployed on an ECS instance in a different region.
  - Some modules of the application are deployed on an ECS instance in the same region with the same *network type*, while other modules are deployed in systems other than Alibaba Cloud.

### Procedure

1. Log on to the [HybridDB for PostgreSQL console](#).
2. Select the Region of the instance.
3. Locate the target instance. In the Actions column, click Manage.
4. Click Database Connection on the left-side navigation.

5. On the Database Connection page, Click Release Internet Address.

If you haven't applied for an Internet address since you created an instance, there is only Apply for Internet address on the Database Connection page.

6. Click OK on the dialog box to release the Internet address.

#### Related API

API	Description
<a href="#">#unique_16</a>	Release the Internet address of an instance.

## 2.4 Upgrade the instance configuration

During the usage of HybridDB for PostgreSQL, some computing resources (such as CPU, disk space, memory, and the number of data processing nodes) may become the bottleneck hindering further growth of data processing speed as the data size and computing workload surge dynamically.

HybridDB for PostgreSQL provides online upgrading of the instance configuration to support dynamic expansion of instances, but downgrading instance configuration is not supported. This document describes how to upgrade the instance configuration.

#### View the instance configuration

HybridDB for PostgreSQL instance configuration includes [group types](#) and [number of groups](#). For more information, see [Instance types](#).

Follow these steps to view the current instance configuration.

1. Log on to the [HybridDB for PostgreSQL console](#).
2. Select the region of the instance. For example, China East 1.
3. Click Manage on the right side of the target instance.

On the Basic Information page, the Configuration Information section displays the instance class, instance details, instance groups, and the total computing resources.

HybridDB for PostgreSQL currently has two instance classes available:

- High-performance group: the group type name starts with gpdb.group.segsdx. This type features a better I/O capability that secures higher performance.

- **High-capacity group:** the group type name starts with gpdb.group.seghdx. This type features a larger and more affordable space to meet higher storage demands.

### Upgrade the instance configuration

Follow these steps to upgrade the instance configuration.

1. Log on to the [HybridDB for PostgreSQL console](#).
2. Select the region of the instance. For example, China East 1.
3. Click the Upgrade on the right side of the target instance.
4. On the Configuration upgrade page, select the expected group type and group quantity, and then click Activate.

HybridDB for PostgreSQL supports a diversity of group type and group quantity combos. For more information, see [Configuration combo lookup table](#). A new group type and group quantity combo must meet the following constraints:

- The new and old computing groups must be of the same instance class, and the new group configuration must be equal to or higher than the old one.
- If the new group configuration is equal to the old group type, the new group quantity must be larger than the old one.

Apart from the preceding constraints, you must also evaluate the data size and computing workload of your service to select a proper group type and quantity combo. For more information, see [How to select instance type](#).



#### Note:

The instance upgrading process may take 30 minutes to three hours depending on your data size. Your instance remains read-only in this process to ensure data consistency, and experiences two transient disconnections. Be prepared in advance. When the upgrading process is completed, the instance resumes the running state. You can visit the database normally and the instance's database kernel is automatically upgraded to the latest version.

After the preceding operations are done, you can return to the console to check the running state of the target instance. When the upgrading process is completed, the instance state becomes Running. Otherwise, it is in Upgrading.



## 2.5 Change connection addresses

In HybridDB for PostgreSQL, you can change the connection address of an instance . For example, if you switch your service to a different HybridDB for PostgreSQL instance, you do not need to modify the application. You only need to configure the new instance to use the connection address of the old instance.

### Prerequisites

- The instance is running.
- The instance has set up a whitelist.

### Procedure

1. Log on to the [HybridDB for PostgreSQL console](#).
2. Select the Region of the instance.
3. Locate the target instance. In the Actions column, click Manage.
4. In the left-side navigation pane, click Database Connection.
5. On the Database Connection page, click Modify Connection Address.
6. In the dialog box displayed, click Connection Type and select a network type.

You can select Intranet Address or Internet Address. The Internet Address option is only available after you have applied for a public IP address.

7. Enter the relevant information in Connection Address and Port, and click OK.

After the page is refreshed, the new connection address is displayed.

### Related API

API	Description
<a href="#">#unique_22</a>	Modifies the connection address and the port for an instance.

## 2.6 Restart an instance

HybridDB for PostgreSQL keeps updating the database kernel version to better meet your requirements. The latest database kernel version is used by default when you create an instance. After a new version is released, you can restart the instance to update its database kernel to enjoy new features in the new version. This document describes how to restart an instance.

## Procedure

Follow these steps to restart an instance.

1. Log on to the [HybridDB for PostgreSQL console](#).
2. Select the region of the instance. For example, China East 1.
3. Click Manage on the right side of the target instance to go to the Basic Information page.
4. Click Restart Instance on the upper right corner of the page and click OK in the dialog box. If you have your mobile phone associated to the instance, you must verify the operation by using the verification code sent to your mobile phone.



### Note:

The restart process usually takes 3 to 30 minutes. During this period, the instance is unavailable for external services. Be prepared in advance. After the instance is restarted, the instance resumes the running state and you can visit the database normally.

After the preceding operations are done, you can return to the console to check the running state of the target instance. When the instance restart is completed, the instance state becomes Running. Otherwise, it is in Restarting.

## Related API

API	Description
<a href="#">#unique_24</a>	Restart an instance.

## 2.7 Release an instance

You can manually release Pay-As-You-Go instances based on your business needs.

### Prerequisite

The instance must be a Pay-As-You-Go type instance.



### Note:

Pay-As-You-Go instances can be released at any time.

## Procedure

1. Log on to the [HybridDB for PostgreSQL console](#).

2. Select the region of the instance you want to release.
3. Select the instance and click Manage in the Actions bar.
4. On the Basic Information page, click Release on the right-side of the Status section.
5. In the dialog box that appears, click the check box before Yes, delete this instance and then click OK to release the specified instance.

**Notice:**

Released instances cannot be recovered. Make sure if you need the instance before you perform this operation.

**Related API**

API	Description
<a href="#">#unique_26</a>	Release an instance.

## 3 Migrate data

---

For more information about operations on data migration, see [Migrate data using different solutions](#).

## 4 User and permission management

You can do the [user management](#) and [permission management](#) to secure your HybridDB for PostgreSQL databases. This documents describes the corresponding methods.

### User management

During an instance creation, the system requires you to specify an initial user name and password. This initial user is the “root user”. After the instance is ready, you can connect to the database with this root user account. You can view the information of all the users by running the `\du+` command after you connect to the database using [psql](#) (a client tool of PostgreSQL or Greenplum). Refer to the following example:



#### Note:

Some other internal management users will also be created apart from the root user.

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

HybridDB for PostgreSQL does not enable the super user permission, but offers a corresponding role of RDS\_SUPERUSER, which is consistent with the permission system in ApsaraDB for RDS (PostgreSQL). Therefore, the root user (such as the `root_user` in the preceding example) has the RDS\_SUPERUSER permission. You can only identify this permission attribute by viewing the user description.

The root user has the following permissions:

- CREATEROLE, CREATEDB and LOGIN permissions, not including the SUPERUSER permission. You can use the root user account to create databases and users.
- View and modify the data tables of other non-super users and perform actions such as SELECT, UPDATE, DELETE, and changing Owner.
- View the connection information of other non-super users, cancel the SQL statement, and terminate the connection.
- Run the CREATE EXTENSION and DROP EXTENSION commands to create and delete extensions.

- Create other users with the RDS\_SUPERUSER permission. For example,

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

## Permission management

You can manage permissions at the database, schema and table levels. For example , if you want to grant the reading permission of a table to a user and revoke the modification permission, you can use the following example:

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

## Reference

For more specific user and permission management methods, see [Managing Roles and Privileges](#).

## 5 Extension management

---

HybridDB for PostgreSQL is developed based on the Greenplum Database and is enhanced with some in-depth extensions by Alibaba Cloud. This document introduces the *extension types*, and how to *create* or *delete* an extension.

### Extension types

HybridDB for PostgreSQL supports the following extensions:

- **PostGIS:** supports geographic information data.
- **MADlib:** supports function library on Machine Learning.
- **fuzzystrmatch:** supports fuzzy matching of strings.
- **orafunc:** supports some Oracle functions.
- **oss\_ext:** supports reading data from OSS.
- **hll:** supports using the HyperLogLog algorithm to perform statistical analysis.
- **pljava:** supports compiling user-defined functions (UDF) in PL/Java.
- **pgcrypto:** supports encryption functions.
- **intarray:** supports integer array-related functions, operators and indexes.

### Create an extension

Run the following command to create an extension:

```
CREATE EXTENSION <extension name>;  
CREATE SCHEMA <schema name>;  
CREATE EXTENSION IF NOT EXISTS <extension name> WITH SCHEMA <schema name>;
```



#### Note:

You need to create the **plpythonu** extension before creating the **MADlib** extension, as shown in the following example.

```
CREATE EXTENSION plpythonu;  
CREATE EXTENSION madlib;
```

### Delete an extension

Run the following command to delete an extension:



#### Note:

**If some other objects are dependent on the extension, you need to add the CASCADE key word to remove all the dependencies first.**

```
DROP EXTENSION <extension name>;  
DROP EXTENSION IF EXISTS <extension name> CASCADE;
```



## 6 Operations of JSON data

---

The JSON type has become the standard data type of the Internet and the Internet of Things (IoT). You can view the specific protocols at the [JSON Official Website](#). PostgreSQL supports JSON well. HybridDB for PostgreSQL also supports the JSON data type based on the PostgreSQL syntax.

This document introduces the basic operations and supported objects of the JSON data in HybridDB for PostgreSQL, including [checking compatibility](#), [converting strings to JSON](#), [internal data types](#), [operators](#), and [functions](#). In addition, some [usage examples](#) are provided for your reference.

Check whether the current version supports JSON

Start a HybridDB for PostgreSQL instance, and run the following command to check whether the current version supports JSON or not:

```
=> SELECT '""'::json;
```

If the operation fails, restart the instance and run the preceding command again.

This command dictates a force type conversion from a string to the JSON format, and the following results indicates whether the JSON type is supported.

- If the system prompts the following response, it indicates that the JSON type is supported and the instance is ready for use.

```
json
-----
""
(1 row)
```

- If the system prompts the following response, it indicates that the JSON type is not supported yet.

```
ERROR:  type "json" does not exist
LINE 1: SELECT '""'::json;
                  ^
```

JSON conversion in the database

Database operations mainly involve: read and write. Writing JSON data means converting strings to JSON format. The content in the strings must conform to the JSON standard, including strings, numbers, arrays, and objects. For example:

## String

```
=> SELECT '"hijson"'::json;
 json
-----
 "hijson"
(1 row)
```

`::` represents force type conversion in PostgreSQL, Greenplum and HybridDB for PostgreSQL. The JSON type input function is called during the conversion process. Therefore, JSON format check is performed during the type conversion as follows:

```
=> SELECT '{hijson:1024}'::json;
ERROR:  invalid input syntax for type json
LINE 1: SELECT '{hijson:1024}'::json;
                  ^
DETAIL:  Token "hijson" is invalid.
CONTEXT:  JSON data, line 1: {hijson...
=>
```

The aforementioned `"` are necessary for `"hijson"`. Because the JSON standard requires the KEY value to be a string, the `{hijson:1024}` here returns a syntax error.

Apart from the type conversion, the conversion from the database record to the JSON string is also performed.

We do not normally use only one string or one number for JSON, but an object that contains one or more key-value pairs. So for Greenplum, conversion to objects is applicable to a majority of JSON scenarios, such as:

```
=> select row_to_json(row('{"a":"a"}', 'b'));
 row_to_json
-----
 {"f1":"{\\"a\\":\\"a\\"}", "f2":"b"}
(1 row)
=> select row_to_json(row('{"a":"a"}'::json, 'b'));
 row_to_json
-----
 {"f1":{"a":"a"}, "f2":"b"}
(1 row)
```

We can also see the differences between the string and JSON here, so as to conveniently convert a full record into the JSON type.

## JSON internal data types

### · Object

The object is the most frequently used data in JSON, such as:

```
=> select '{"key":"value"}'::json;
 json
```

```
-----
{"key":"value"}
(1 row)
```

- Integer and floating point

The JSON protocol only has three types of numbers: integer, floating point and constant expression. Greenplum provides good support for all three number types.

```
=> SELECT '1024'::json;
      json
-----
    1024
(1 row)
=> SELECT '0.1'::json;
      json
-----
     0.1
(1 row)
```

The following information is required in some special situations:

```
=> SELECT '1e100'::json;
      json
-----
    1e100
(1 row)
=> SELECT '{"f":1e100}'::json;
      json
-----
{"f":1e100}
(1 row)
```

And the extra-long number is also included as follows:

```
=> SELECT '9223372036854775808'::json;
      json
-----
9223372036854775808
(1 row)
```

- Array

```
=> SELECT '[[1,2], [3,4,5]]'::json;
      json
-----
[[1,2], [3,4,5]]
(1 row)
```

## Operators

### Operator types supported by JSON

```
=> select oprname,oprcode from pg_operator where oprleft = 3114;
oprname |      oprcode
-----+-----
->      | json_object_field
->>     | json_object_field_text
```

```

->      | json_array_element
->>     | json_array_element_text
#>      | json_extract_path_op
#>>     | json_extract_path_text_op
(6 rows)

```

## Basic usage

```

=> SELECT '{"f":"1e100"}'::json -> 'f';
?column?
-----
"1e100"
(1 row)
=> SELECT '{"f":"1e100"}'::json ->> 'f';
?column?
-----
1e100
(1 row)
=> select '{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}'::json#>array
['f4','f6'];
?column?
-----
"stringy"
(1 row)
=> select '{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}'::json#>'f4,
f6';
?column?
-----
"stringy"
(1 row)
=> select '{"f2":["f3",1],"f4":{"f5":99,"f6":"stringy"}}'::json#>>'f2,
0';
?column?
-----
f3
(1 row)

```

## JSON functions

### Supported functions

```
postgres=# \df *json*
```

List of functions				
Schema	Name	Result data type	Argument data types	
				Type
pg_catalog	array_to_json	json		anyarray
pg_catalog	array_to_json, boolean	json		anyarray
pg_catalog	json_array_element	json	integer	from_json
pg_catalog	json_array_element_text	text	integer	from_json
pg_catalog	json_array_elements	SETOF json	json	from_json
pg_catalog	json_array_length	integer	json	json

pg_catalog   json_each	SETOF record	from_json
json, OUT key text, OUT value json	normal	
pg_catalog   json_each_text	SETOF record	from_json
json, OUT key text, OUT value text	normal	
pg_catalog   json_extract_path	json	from_json
json, VARIADIC path_elems text[]	normal	
pg_catalog   json_extract_path_op	json	from_json
json, path_elems text[]	normal	
pg_catalog   json_extract_path_text	text	from_json
json, VARIADIC path_elems text[]	normal	
pg_catalog   json_extract_path_text_op	text	from_json
json, path_elems text[]	normal	
pg_catalog   json_in	json	cstring
	normal	
pg_catalog   json_object_field	json	from_json
json, field_name text	normal	
pg_catalog   json_object_field_text	text	from_json
json, field_name text	normal	
pg_catalog   json_object_keys	SETOF text	json
	normal	
pg_catalog   json_out	cstring	json
	normal	
pg_catalog   json_populate_record	anyelement	base
anyelement, from_json json, use_json_as_text boolean	normal	
pg_catalog   json_populate_recordset	SETOF anyelement	base
anyelement, from_json json, use_json_as_text boolean	normal	
pg_catalog   json_recv	json	internal
	normal	
pg_catalog   json_send	bytea	json
	normal	
pg_catalog   row_to_json	json	record
	normal	
pg_catalog   row_to_json	json	record,
boolean	normal	
pg_catalog   to_json	json	
anyelement		normal
(24 rows)		

## Basic usage

```
=> SELECT array_to_json('{{1,5},{99,100}}'::int[]);
array_to_json
-----
[[1,5],[99,100]]
(1 row)
=> SELECT row_to_json(row(1,'foo'));
row_to_json
-----
{"f1":1,"f2":"foo"}
(1 row)
=> SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]');
json_array_length
-----
5
(1 row)
=> select * from json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null,"f5":99,"f6":"stringy"}') q;
key | value
-----+-----
f1 | [1,2,3]
f2 | {"f3":1}
f4 | null
f5 | 99
```

```

f6 | "stringy"
(5 rows)
=> select json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null,"f5":
null}');
      json_each_text
-----
(f1,"[1,2,3]")
(f2,"{"f3":1}")
(f4,)
(f5,null)
(4 rows)
=> select json_array_elements('[1,true,[1,[2,3]],null,{"f1":1,"f2":[7,
8,9]},false]');
      json_array_elements
-----
1
true
[1,[2,3]]
null
{"f1":1,"f2":[7,8,9]}
false
(6 rows)
create type jpop as (a text, b int, c timestamp);
=> select * from json_populate_record(null::jpop,'{"a":"blurfl","x":43
.2}', false) q;
   a   | b | c
-----+---+---
blurfl |   | 
(1 row)
=> select * from json_populate_recordset(null::jpop,'[{"a":"blurfl
","x":43.2},{ "b":3,"c":"2012-01-20 10:42:53"}]',false) q;
   a   | b | c
-----+---+-----
blurfl | 3 | Fri Jan 20 10:42:53 2012
(2 rows)

```

## Code Examples

### Create a table

```

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

```

(2 rows)

## Multi-table JOIN

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

## JSON function indexing

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



### Note:

JSON type cannot be used as the distribution key for now and the JSON aggregate functions are not supported.

The following is an example of Python access:

```
#!/bin/env python
import time
import json
import psycopg2
def gpquery(sql):
    conn = None
```

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



## 7 Use HyperLogLog

---

HybridDB for PostgreSQL is nested with native features of Greenplum Database, and also supports HyperLogLog. It provides solutions for industries with the Internet advertisement analysis requirements and requirements similar to estimation analysis computing to facilitate quick estimation of PV, UV, and other business metrics.

### Create a HyperLogLog extension

Run the following command to create a HyperLogLog extension:

```
CREATE EXTENSION hll;
```

### Basic types

- Run the following command to create a table containing the hll field:

```
create table agg (id int primary key,userid hll);
```

- Run the following command to convert int to hll\_hashval:

```
select 1::hll_hashval;
```

### Basic operators

- The hll type supports =, !=, <>, ||, and #.

```
select hll_add_agg(1::hll_hashval) = hll_add_agg(2::hll_hashval);  
select hll_add_agg(1::hll_hashval) || hll_add_agg(2::hll_hashval);  
select #hll_add_agg(1::hll_hashval);
```

- The hll\_hashval type supports =, !=, and <>.

```
select 1::hll_hashval = 2::hll_hashval;  
select 1::hll_hashval <> 2::hll_hashval;
```

### Basic functions

- The supported functions include hll\_hash\_boolean, hll\_hash\_smallint, hll\_hash\_bigint, and other hash functions.

```
select hll_hash_boolean(true);
```

```
select hll_hash_integer(1);
```

- **hll\_add\_agg:** Used to convert int to the hll format.

```
select hll_add_agg(1::hll_hashval);
```

- **hll\_union:** The union of hll.

```
select hll_union(hll_add_agg(1::hll_hashval),hll_add_agg(2::hll_hashval));
```

- **hll\_set\_defaults:** Used to set the precision.

```
select hll_set_defaults(15,5,-1,1);
```

- **hll\_print:** Used for debug information.

```
select hll_print(hll_add_agg(1::hll_hashval));
```

## Examples

```
create table access_date (acc_date date unique, userids hll);
insert into access_date select current_date, hll_add_agg(hll_hash_integer(user_id)) from generate_series(1,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_integer(user_id)) from generate_series(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_integer(user_id)) from generate_series(9000,40000) t(user_id);
postgres=# select #userids from access_date where acc_date=current_date;
           ?column?
-----
9725.85273370708
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-1;
           ?column?
-----
14968.6596883279
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-2;
           ?column?
-----
29361.5209149911
(1 row)
```

## 8 Use the create Library command

---

HybridDB for PostgreSQL introduces the “Create Library/Drop Library” command to allow you to import custom software packages. For PL/Java UDF examples created by using this command, see [PL/Java UDF Usage](#).

This document describes the usage of the Create/Drop Library command.

### Syntax

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

### Parameter description:

- **library\_name**: name of the library to be installed. If the name of the library to be installed conflicts with an existing library's name, the existing library must be deleted first to install the new one.
- **LANGUAGE [JAVA]**: the language to use. Currently only PL/Java is supported.
- **oss\_location**: location of the package file. You can specify the OSS bucket and object name. Only one object can be specified and the specified object must not be a compressed file. The format is:

```
oss://oss_endpoint filepath=[folder/[folder/]...]/file_name id=
userossid key=userosskey bucket=ossbucket
```

- **file\_content\_hex**: file content. The byte stream is in hexadecimal notation. For example, 73656c6563742031 indicates the hexadecimal byte stream of “select 1” . With this syntax, you can directly import package files without using the OSS.
- **ownername**: specify the user.
- **DROP LIBRARY**: delete a library.

## Examples

- Install a JAR package named `analytics.jar`.

```
create library example language java from 'oss://oss-cn-hangzhou.
aliyuncs.com filepath=analytics.jar id=xxx key=yyy bucket=zzz';
```

- Import the file content directly and the byte stream is in hexadecimal notation.

```
create library pglib LANGUAGE java VALUES '73656c6563742031' OWNER
"myuser";
```

- Delete a library.

```
drop library example;
```

- View installed libraries.

```
select name, lanname from pg_library;
```

## 9 Use PL / Java UDF

---

HybridDB for PostgreSQL supports compiling and uploading JAR software packages written in PL/Java languages, and using these JAR packages to create user-defined functions (UDF). The PL/Java language supported in this feature is Community Edition PL/Java 1.5.0 and the JVM version is 1.8.

This document describes how to create a PL/Java UDF. For more PL/Java examples, see [PL/Java Code](#). You can also view [How to Compile](#).

### Procedure

1. In HybridDB for PostgreSQL, run the following command to create a PL/Java plug-in. The command only needs to be ran once for the database.

```
create extension pljava;
```

2. Compile the UDF based on your business requirements. For example, you can use the following code to compile the `Test.java` file:

```
public class Test
{
    public static String substring(String text, int beginIndex,
                                   int endIndex)
    {
        return text.substring(beginIndex, endIndex);
    }
}
```

3. Compile the `manifest.txt` file.

```
Manifest-Version: 1.0
Main-Class: Test
Specification-Title: "Test"
Specification-Version: "1.0"
Created-By: 1.7.0_99
Build-Date: 01/20/2016 21:00 AM
```

4. Run the following command to compile and package the program.

```
javac Test.java
```

```
jar cfm analytics.jar manifest.txt Test.class
```

5. Upload the `analytics.jar` file generated in Step 4 to the OSS by using the following OSS console command.

```
osscmd put analytics.jar oss://zzz
```

6. In HybridDB for PostgreSQL, run the “Create Library” command to import the file to HybridDB for PostgreSQL.

**Note:**

The `Create Library` command only supports filepath and you can import one file a time. In addition, the “Create Library” command also supports byte streams to directly import files without using the OSS. For more information, see [Use the Create Library Command](#).

```
create library example language java from 'oss://oss-cn-hangzhou.
aliyuncs.com filepath=analytics.jar id=xxx key=yyy bucket=zzz';
```

7. In HybridDB for PostgreSQL, run the following command to create and use the UDF .

```
create table temp (a varchar) distributed randomly;
insert into temp values ('my string');
create or replace function java_substring(varchar, int, int) returns
varchar as 'Test.substring' language java;
select java_substring(a, 1, 5) from temp;
```

## 10 Use SortKey

SortKey is a table attribute that helps store data in the order of the SortKey to files on a disk. SortKey offers the following advantages:

- Speed up column-store optimization. The min and max meta information it collects seldom overlaps with each other, featuring good filtering friendliness.
- Avoid sorting SQL data containing “order by” and “group by” for a second time. The data directly read from the disk is ordered as required by the sorting conditions.

This document describes the usage of SortKey in different use cases.

### Define the SortKey

You can use the `CREATE TABLE` command to define a new table containing a SortKey.

The syntax is as follows:

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

```
)
```

### Example

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

### Sort the table

The command is as follows:

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

### Change the SortKey

The command is as follows:

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



#### Note:

This command only changes catalog without sorting the data immediately. To sort data, you must use the `VACUUM SORT ONLY` command.

### Example

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



#### Note:

After you update a table (such as Insert, Update, and Delete), data in the table is deemed as non-SortKey-ordered. Data read from the disk for a query is not regarded as ordered. In this case, you must rerun the `VACUUM SORT ONLY` command to re-organize data in the table.