FROM INTRODUCTION TO PRACTICE

Lesson 7: OceanBase's Extended Functions under Mysql Tenants

Peng Wang
OceanBase Global Technical Evangelist

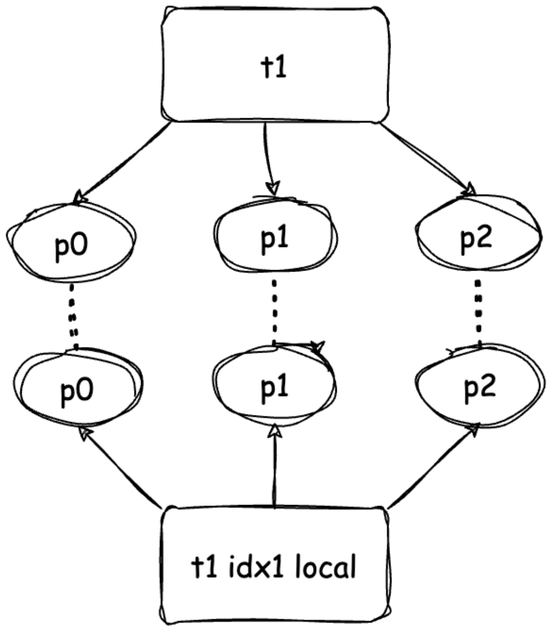
Agenda

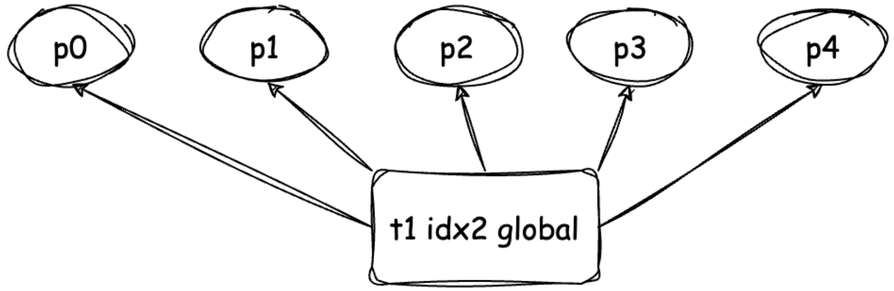
- Global Index
- Recycle Bin
- Tablegroup
- Sequence
- Self-study Content

Global Index

Function Definition

```
CREATE TABLE t1(c1 int, c2 int)
    PARTITION BY RANGE(c1)(
        PARTITION p0 VALUES LESS THAN(100),
        PARTITION p1 VALUES LESS THAN(200),
        PARTITION p2 VALUES LESS THAN MAXVALUE);
```






```
create index idx1 on t1(c1) local;
```

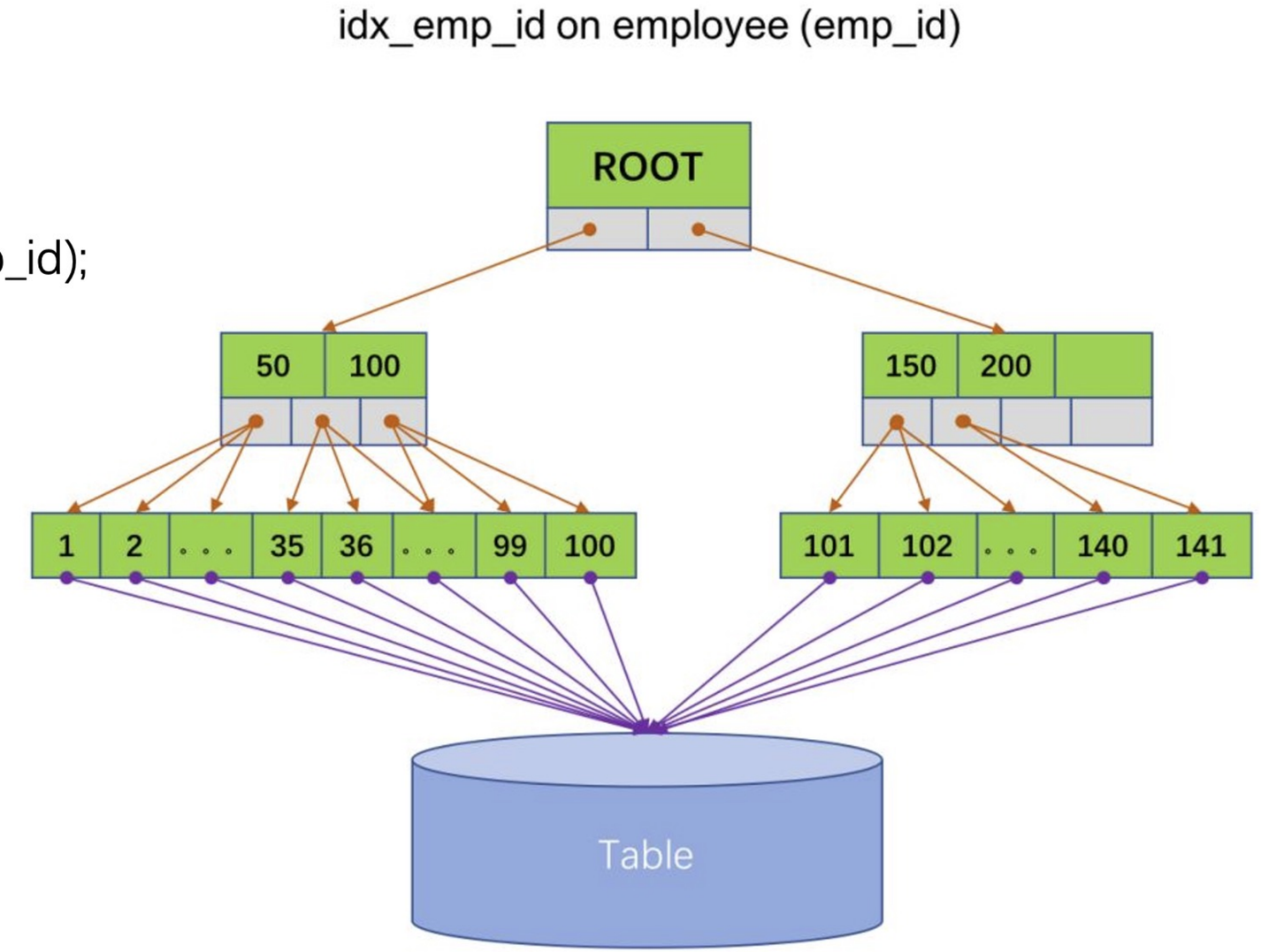
```
create index idx2 on t1(c1) global
    PARTITION BY RANGE(c1)(
        PARTITION p0 VALUES LESS THAN(10),
        PARTITION p1 VALUES LESS THAN(20),
        PARTITION p2 VALUES LESS THAN(30),
        PARTITION p3 VALUES LESS THAN(40),
        PARTITION p4 VALUES LESS THAN MAXVALUE);
```

Global Index

Reason for Introduction

CREATE INDEX idx_emp_id ON employee(emp_id);

- B+ Tree structure
- One-to-one correspondence between key values and data
- Leaf nodes are sorted by key value
- The leaf node points to the indexed main table data



Global Index

Reason for Introduction

Grammar:

```
CREATE INDEX idx_emp_name ON
    employ(emp_name) LOCAL;
```

Features:

- One-to-one correspondence with the sharding information of the main table
- Each subtree only indexes data within the shard

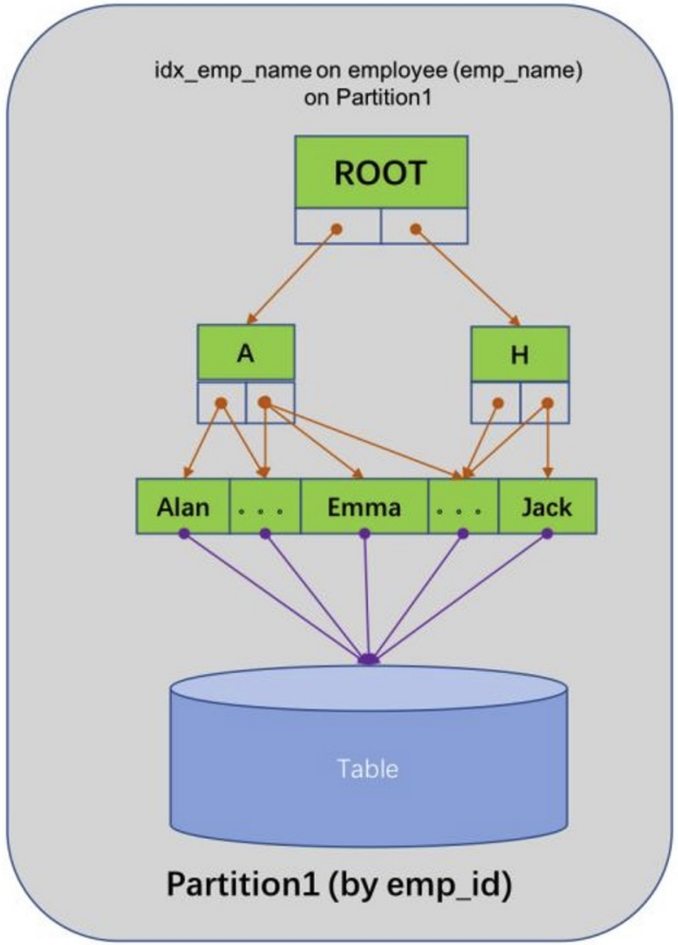
Disadvantage:

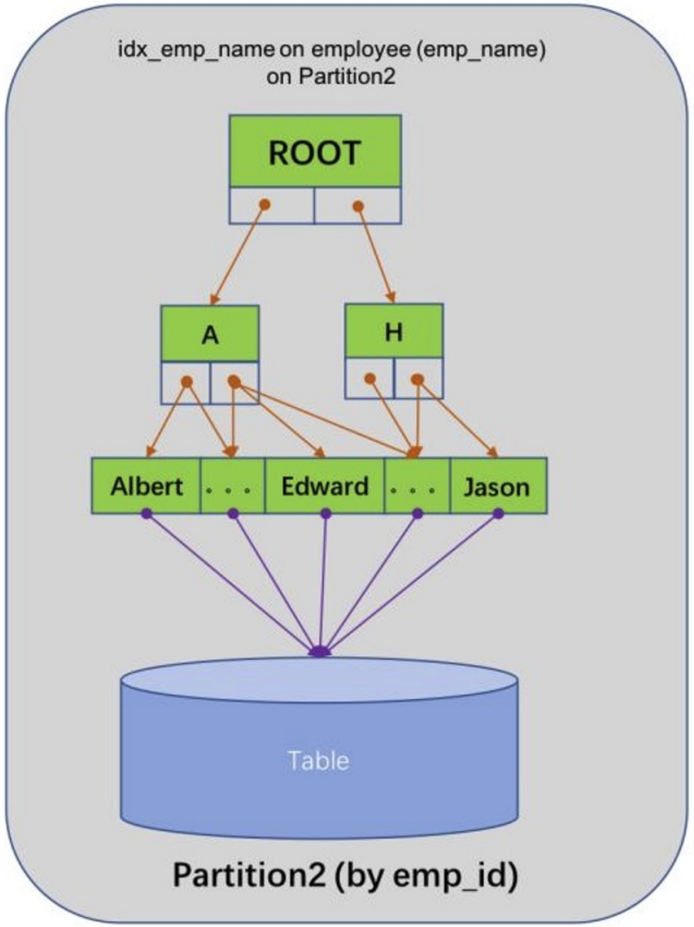
- The query requires the specified partition key

```
select * from employee where
        emp_name='Edward';
```

```
| ============================================================
|ID|OPERATOR              |NAME                  |EST. ROWS|COST |
------------------------------------------------------------------
|0 |PX COORDINATOR        |                      |9900     |62541|
|1 | EXCHANGE OUT DISTR   |:EX10000              |9900     |61018|
|2 |  PX PARTITION ITERATOR|                     |9900     |61018|
|3 |   TABLE SCAN         |employee(idx_emp_name)|9900     |61018|
============================================================

Outputs & filters:
-------------------------------------
  0 - output([employee.emp_id], [employee.emp_name], [employee.dpet_id]), filter(nil)
  1 - output([employee.emp_id], [employee.emp_name], [employee.dpet_id]), filter(nil), dop=1
  2 - output([employee.emp_id], [employee.emp_name], [employee.dpet_id]), filter(nil)
  3 - output([employee.emp_id], [employee.emp_name], [employee.dpet_id]), filter(nil),
      access([employee.emp_id], [employee.emp_name], [employee.dpet_id]), partitions(p[0-9])
 |
```





Global Index

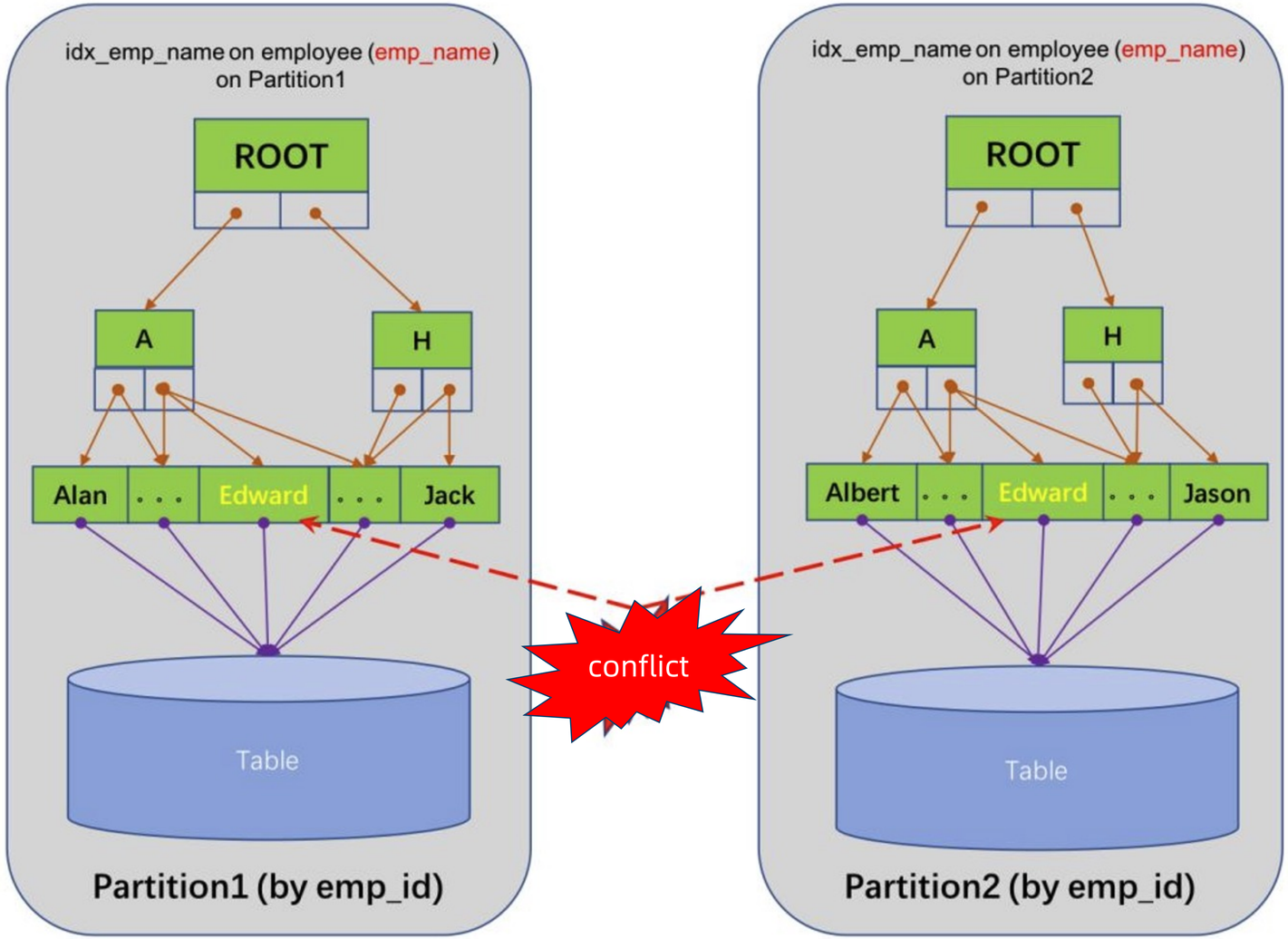
Reason for Introduction

Disadvantage:

- Global uniqueness cannot be guaranteed

```
CREATE UNIQUE INDEX idx_emp_name
 ON employ(emp_name) LOCAL;
```

```
ERROR 1503 (HY000): A UNIQUE INDEX
must include all columns in the
table's partitioning function
```



Global Index

Reason for Introduction

Grammar:

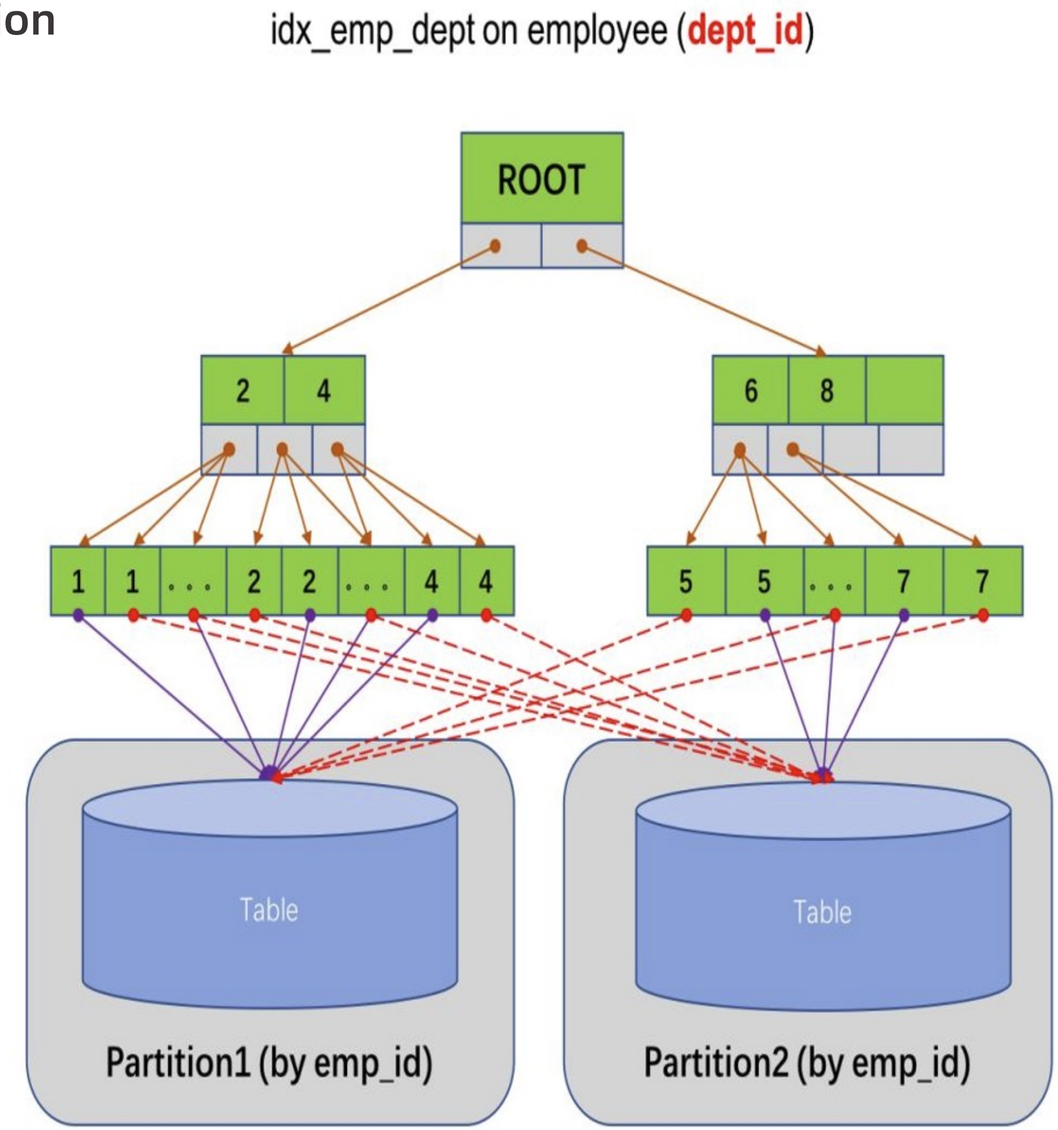
```
CREATE INDEX idx_emp_dept ON
    employ(dept_id) GLOBAL;
```

Features:

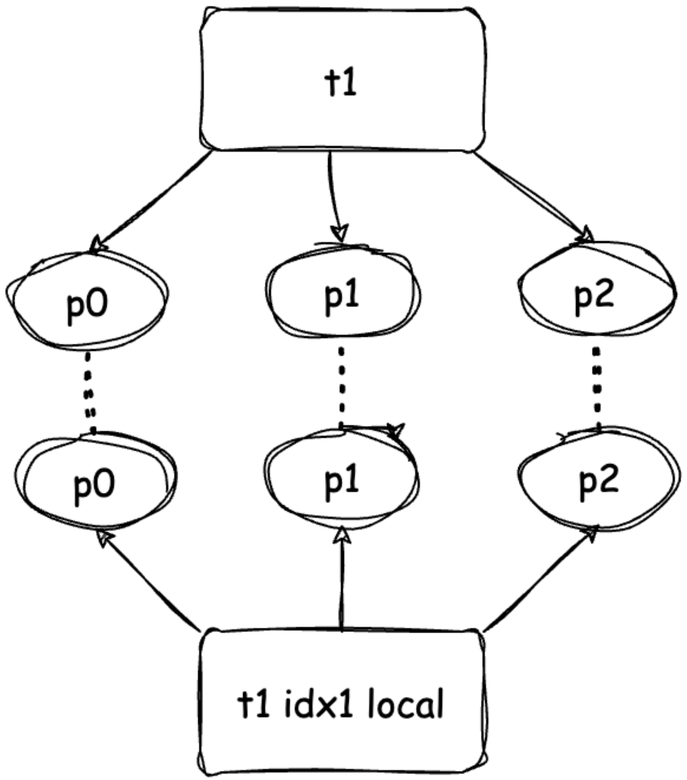
- The shard information of the index and the main table are independent of each other
- A key value may correspond to data in multiple main table shards

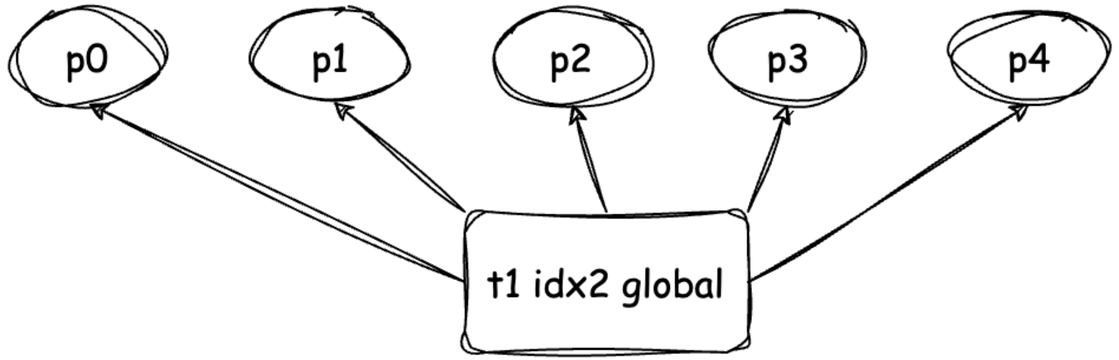
Advantages:

- Index retrieval does not require specifying a partition key
- The global uniqueness of the key value can be guaranteed



```
CREATE TABLE t1(c1 int, c2 int)
    PARTITION BY RANGE(c1)(
        PARTITION p0 VALUES LESS THAN(100),
        PARTITION p1 VALUES LESS THAN(200),
        PARTITION p2 VALUES LESS THAN MAXVALUE);
```






```
create index idx1 on t1(c1) local;
```

```
create index idx2 on t1(c1) global
    PARTITION BY RANGE(c1)(
        PARTITION p0 VALUES LESS THAN(10),
        PARTITION p1 VALUES LESS THAN(20),
        PARTITION p2 VALUES LESS THAN(30),
        PARTITION p3 VALUES LESS THAN(40),
        PARTITION p4 VALUES LESS THAN MAXVALUE);
```

Global Index

Reason for Introduction

Grammar:

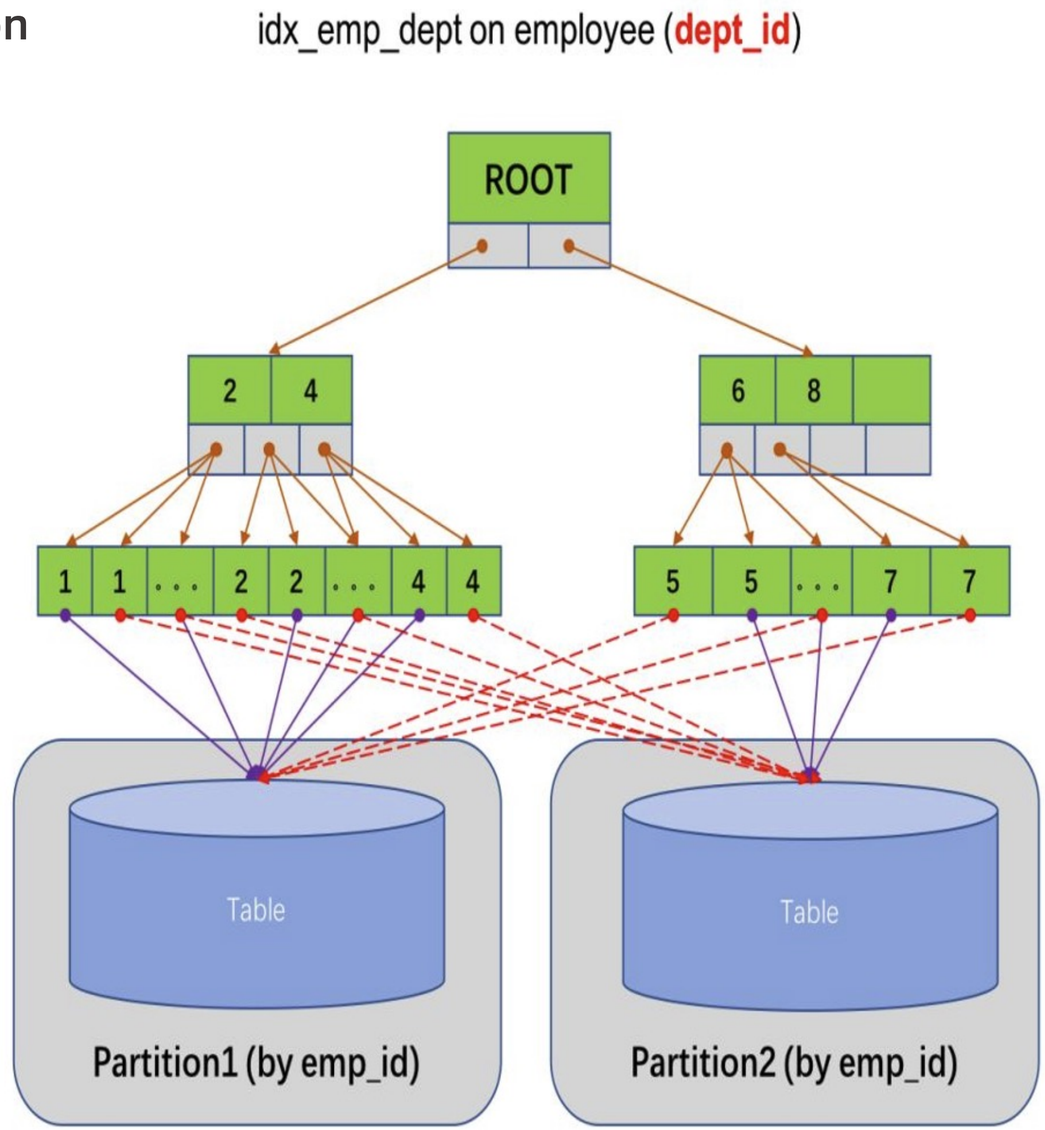
```
CREATE INDEX idx_emp_dept ON
    employ(dept_id) GLOBAL;
```

Features:

- The shard information of the index and the main table are independent of each other
- A key value may correspond to data in multiple main table shards

Advantages:

- Index retrieval does not require specifying a partition key
- The global uniqueness of the key value can be guaranteed



Global Index

Related Syntax

```
-- Create a table
create table t1(
  c1 int primary key,
  c2 int
  ) partition by hash(c1) partitions 5;
-- Create a partition index with a different partitioning method from the main table
create index g_idx on t1(c2) global
  partition by range(c2)
    (partition p0 values less than(100),
     partition p1 values less than(200),
     partition p2 values less than(300)
    );
```

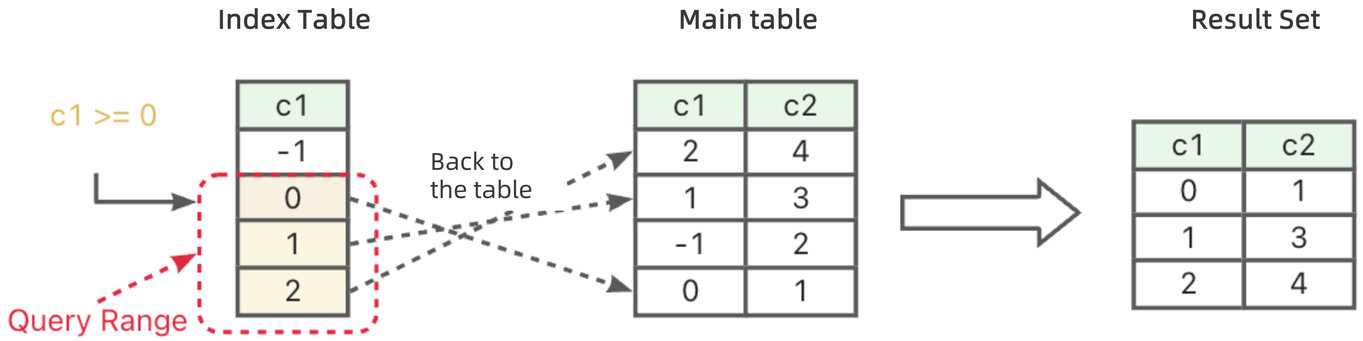
Equivalent

```
-- The above two SQL statements are equivalent to this one SQL statement
CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `g_idx` (`c2`) GLOBAL
   partition by range(c2) -- Index partitioning
    (partition `p0` values less than (100),
     partition `p1` values less than (200),
     partition `p2` values less than (300)
    )
  ) partition by hash(c1) -- Partitioning method of the main table
    (partition `p0`,
     partition `p1`,
     partition `p2`,
     partition `p3`,
     partition `p4`);
```

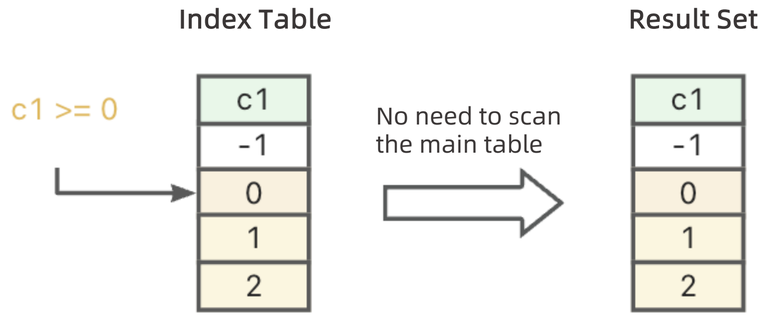
Global Index

Applicable Scenarios & FAQs

```
create table t1(c1 int, c2 int, index idx(c1));
select /*+index(t1 idx) */ c1, c2 from t1 where c1 >= 0;
```




```
select /*+ index(t1 idx) */ c1 from t1 where c1 >= 0;
```



Global Index

Applicable Scenarios & FAQs

Applicable Scenarios:

- The main table is a partitioned table, but the index key does not contain the main table partition key.
- If you need to ensure that the index key satisfies the uniqueness constraint and the index key does not contain the main table partition key information.

Global Index

Applicable Scenarios & FAQs

FAQs:

- Why can't the primary key index be set as a global attribute?
- Why must the partition key of a partitioned table be included in the local unique index and primary key index?

```
obclient [test]> create table t1(c1 int, c2 int);
Query OK, 0 rows affected (0.414 sec)

obclient [test]> select c.table_id, column_id, column_name, rowkey_position, is_hidden from oceanbase.__all_table t, oceanbase.__all_column c
    ->     where t.table_name = 't1' and t.table_id = c.table_id order by column_id;
+----------+-----------+----------------+-----------------+-----------+
| table_id | column_id | column_name    | rowkey_position | is_hidden |
+----------+-----------+----------------+-----------------+-----------+
|   500110 |         1 | __pk_increment |               1 |         1 |
|   500110 |        16 | c1             |               0 |         0 |
|   500110 |        17 | c2             |               0 |         0 |
+----------+-----------+----------------+-----------------+-----------+
3 rows in set (0.115 sec)
```

Global Index

Applicable Scenarios & FAQs

FAQs:

- Why can't the primary key index be set as a global attribute?
- Why must the partition key of a partitioned table be included in the local unique index and primary key index?

```
create table t1(c1 int unique key, c2 int) partition by hash(c2);
ERROR 1503 (HY000): A UNIQUE INDEX must include all columns in the table's partitioning function

create table t1(c1 int primary key, c2 int) partition by hash(c2);
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

Global Index

Applicable Scenarios & FAQs

When introducing the Global Index, I mentioned that the uniqueness check of data on the unique index is only performed on the current partition. If the unique index does not contain all partition keys, for example, if create table t1(c1 int unique key, c2 int) partition by hash(c2) is executed successfully, the data on the main table may be:

c1	c2
1	1
1	2
2	1
2	2
3	1
3	2

Global Index

At this time, the data of the first partition is: (c1 satisfies the uniqueness in this partition, and the uniqueness check will succeed)

c1	c2
1	1
2	1
3	1

The data of the second partition is: (c1 also satisfies the uniqueness in this partition, and the uniqueness check will also succeed)

c1	c2
1	2
2	2
3	2

Agenda

- Global Index
- Recycle Bin
- Tablegroup
- Sequence
- Self-study Content

Recycle Bin

Function Definition

The recycle bin is a function that stores database objects deleted by users.

In Oceanbase, objects that can enter the recycle bin include databases, tables, indexes, and tenants.

After the information deleted by users is placed in the recycle bin, it still occupies physical space unless it is manually purged (PURGE) or the database is configured to regularly purge the recycle bin objects. Before the recycle bin objects are purged, they can be restored through commands.

Recycle Bin

Reason for Introduction

The function of the recycle bin is to prevent users from accidentally deleting database objects and being unable to restore them.

The MySQL database lacks this important function. OceanBase refers to the function and usage of the recycle bin in the Oracle database and also supports the recycle bin function under the MySQL tenant of OceanBase.

Recycle Bin

Related Syntax

-- Enable/disable the recycle bin function. It is disabled by default.
-- Take effect for the current session

```
SET recyclebin = on;
SET recyclebin = off;
SET session recyclebin = on;
SET session recyclebin = off;
```

-- After adding the global keyword, the command to turn on/off the recycle bin function will take effect in the current tenant.
-- After setting, it will be invalid for the current session but will take effect for subsequent new sessions

```
SET global recyclebin = on;
SET global recyclebin = off;
```

-- Check the status of the recycle bin switch

```
show variables like 'recyclebin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| recyclebin    | ON    |
+---------------+-------+
```

Recycle Bin

Related Syntax

-- Set the periodic cleanup period for objects in the recycle bin. The default value is 0s, which means that this function is not enabled.
-- This command can only be executed by the sys tenant and is effective for the entire cluster

```
alter system set recyclebin_object_expire_time= '7d';
```

automatically cleaned up

-- Set to 0s to disable this function.

```
ALTER SYSTEM SET recyclebin_object_expire_time = "0s";
```

-- View the recyclebin_object_expire_time parameter in the cluster

```
show parameters like 'recyclebin_object_expire_time'\G
*************************** 1. row ***************************
         zone: zone1
     svr_type: observer
       svr_ip: 1.2.3.4
     svr_port: 12345
         name: recyclebin_object_expire_time
    data_type: NULL
        value: 0s
         info: recyclebin object expire time, default 0 that means auto purge recyclebin off. Range: [0s, +∞)
      section: ROOT_SERVICE
        scope: CLUSTER
       source: DEFAULT
   edit_level: DYNAMIC_EFFECTIVE
default_value: 0s
    isdefault: 1
```

Recycle Bin

Related Syntax

```
-- Create database, table, index
create database test_db;
create table t1(c1 int, c2 int, index idx(c2));

-- Drop database, table, index
drop database test_db;
drop table t1;
```

```
-- View Recycle Bin Objects
-- ORIGINAL_NAME indicates the original name of the Recycle Bin object before it was deleted
-- OBJECT_NAME indicates the new name of the Recycle Bin object after it enters the Recycle Bin, to uniquely identify the same type of objects with the same name in the Recycle Bin
show RECYCLEBIN;
+-------------------------------+-----------------+----------+----------------------------+
| OBJECT_NAME                   | ORIGINAL_NAME   | TYPE     | CREATETIME                 |
+-------------------------------+-----------------+----------+----------------------------+
| __recycle_$_1_1713173706419784 | __idx_500044_idx | INDEX    | 2024-04-15 17:35:06.419840 |
| __recycle_$_1_1713173706464688 | t1              | TABLE    | 2024-04-15 17:35:06.465056 |
| __recycle_$_1_1713173712877712 | test_db         | DATABASE | 2024-04-15 17:35:12.877862 |
+-------------------------------+-----------------+----------+----------------------------+
3 rows in set (0.010 sec)
```

```
-- By directly querying the internal table oceanbase._all_recyclebin, you can see detailed information about the recycle bin object before it is deleted, such as database_id, table_id and others before deletion.
select * from oceanbase.__all_recyclebin;
+----------------------------+-----------+--------------------------------+------+-------------+----------+--------------+------------------+
| gmt_create                 | tenant_id | object_name                    | type | database_id | table_id | tablegroup_id | original_name    |
+----------------------------+-----------+--------------------------------+------+-------------+----------+--------------+------------------+
| 2024-04-15 17:35:06.419840 |         0 | __recycle_$_1_1713173706419784 |    2 |      500001 |   500045 |            -1 | __idx_500044_idx |
| 2024-04-15 17:35:06.465056 |         0 | __recycle_$_1_1713173706464688 |    1 |      500001 |   500044 |            -1 | t1               |
| 2024-04-15 17:35:12.877862 |         0 | __recycle_$_1_1713173712877712 |    4 |      500046 |       -1 |            -1 | test_db          |
+----------------------------+-----------+--------------------------------+------+-------------+----------+--------------+------------------+
3 rows in set (0.006 sec)
```

Recycle Bin

Related Syntax

The following example uses the restore table syntax as follows:

```
FLASHBACK TABLE object_name TO BEFORE DROP [RENAME To new_table_name];
```

```
-- Create a table with the same name as the table that was just deleted
create table t1(c1 int);
Query OK, 0 rows affected (0.267 sec)
```

```
-- To restore a table from the Recycle Bin, you need to go to the original database before the table was deleted. The restored table name defaults to the table name before entering the Recycle Bin. You can modify it by RENAME To new_table_name
-- After executing this statement, the restored table name will be the same as the name before entering the recycle bin, and the indexes that entered the recycle bin together with the table will also be restored.
-- If the name of the table before entering the recycle bin is the same as that of an existing table, the system will report an error.
flashback table __recycle_$_1_1713173706464688 to before drop;
ERROR 1050 (42S01): Table 't1' already exists
```

```
-- Modify by RENAME To new_table_name
flashback table __recycle_$_1_1713173706464688 to before drop rename to old_t1;
Query OK, 0 rows affected (0.106 sec)
```

```
-- You can see that the table in the recycle bin and the index on the table are restored together.
obclient [test]> show RECYCLEBIN;
+-------------------------------+---------------+----------+----------------------------+
| OBJECT_NAME                   | ORIGINAL_NAME | TYPE     | CREATETIME                 |
+-------------------------------+---------------+----------+----------------------------+
| __recycle_$_1_1713173712877712 | test_db       | DATABASE | 2024-04-15 17:35:12.877862 |
+-------------------------------+---------------+----------+----------------------------+
```

Recycle Bin

Related Syntax

Command to clear Recycle Bin objects:

purge object_type object_name;

```
show recyclebin;
+------------------------------+-----------------+----------+----------------------------+
| OBJECT_NAME                  | ORIGINAL_NAME   | TYPE     | CREATETIME                 |
+------------------------------+-----------------+----------+----------------------------+
| __recycle_$_1_1713173712877712 | test_db         | DATABASE | 2024-04-15 17:35:12.877862 |
| __recycle_$_1_1713177112706600 | __idx_500048_idx | INDEX    | 2024-04-15 18:31:52.706664 |
| __recycle_$_1_1713177112725848 | t1              | TABLE    | 2024-04-15 18:31:52.727735 |
+------------------------------+-----------------+----------+----------------------------+
3 rows in set (0.011 sec)
```

-- The syntax for cleaning a specified index is: PURGE INDEX object_name

```
purge index __recycle_$_1_1713177112706600;
```

-- The syntax for cleaning a specified table is: PURGE TABLE object_name

```
purge table __recycle_$_1_1713177112725848;
```

-- The syntax for cleaning up a specified database is: PURGE DATABASE object_name

```
purge database __recycle_$_1_1713173712877712;
```

-- The syntax for clearing a tenant is: PURGE TENANT object_name; (tenant-level objects can only be managed through sys tenant)

-- You can also use purge recyclebin to clean up all objects in the recycle bin at once

purge recyclebin;

-- Display objects in the recycle bin after cleaning

```
show recyclebin;
Empty set (0.011 sec)
```

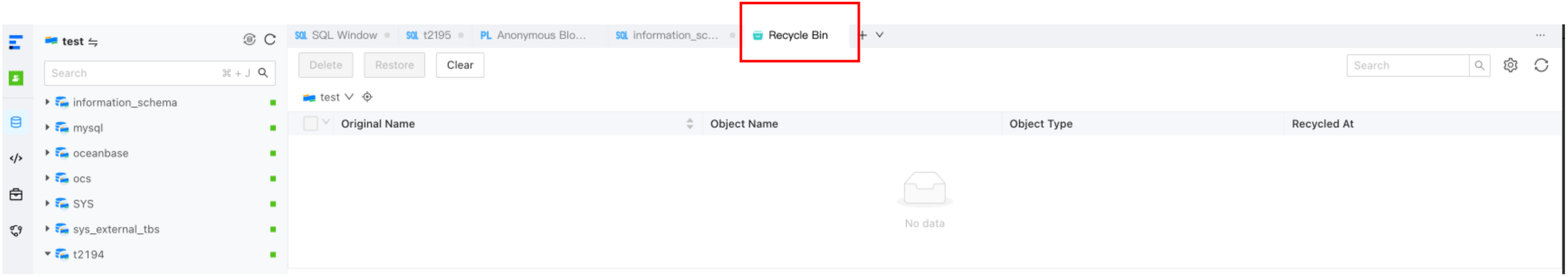
Recycle Bin

Related Syntax

When a special tenant stores important data in a production environment, consider enabling the Recycle Bin function to prevent DBA students from misoperating.

You can appropriately reduce the period of scheduled cleanup of the recycle bin to reduce the additional disk overhead of the data in the Recycle Bin.

The Recycle Bin white screen operation can be achieved through ODC (OceanBase Developer Center).



Recycle Bin

Related Syntax

1. Will disabling the Recycle Bin empty the objects already in the Recycle Bin?

When you disable the recycle bin by setting recyclebin = off, objects already in the recycle bin will not be emptied or affected.

2. Is it supported to flashback and purge recycle bin objects using the original name of the database object?

For table objects, flashback and purge are supported using the original name (ORIGINAL_NAME) before being deleted.

The rules are as follows:

- When a table object with the same original name appears in the recycle bin, a flashback will restore the table that entered the recycle bin the latest. So the operation of flashback original_name can be understood as a stack, where the last table to enter is restored first.
- When a table object with the same original name appears in the recycle bin, purge will clear the table that entered the recycle bin first. Therefore, the operation of purge original_name can be understood as a queue, where the first-in-first-out is emptied.

The above rules refer to the behavior of the Recycle Bin in the Oracle database and are consistent with the behavior of the Oracle Recycle Bin.

For tenant objects, flashback and purge are also supported using the original name (ORIGINAL_NAME) before deletion, following the same rules as above.
For index and database objects, this behavior is not supported.

Recycle Bin

Related Syntax

3. Why don't we support indexes being sent to the Recycle Bin separately?

If you delete an index directly in OceanBase, the index will not be sent to the Recycle Bin. When you delete a table, the index on the table will be sent to the Recycle Bin along with the main table.
Because if you delete an index separately and the deleted index is allowed to go to the Recycle Bin, then when the main table structure or data is changed later, the index in the Recycle Bin has been deleted and cannot be updated synchronously with the main table. Therefore, the index in the Recycle Bin will be inconsistent with the main table data that has not been deleted. Therefore, the deleted index cannot be restored using the flashback command, which undermines the purpose of placing it in the Recycle Bin.

4. Why does the truncate table in version 4.x not go to the recycle bin?

In version 3.x, when the session-level system variable ob_enable_truncate_flashback is set to on, if a truncate table operation is performed by mistake, the table and data before the truncate table operation can be restored through flashback.
In version 4.x, a function regression occurred. The reason is long, but in short, it is caused by a change in the implementation method of truncate table.

Agenda

- Global Index
- Recycle Bin
- Tablegroup
- Sequence
- Self-study Content

Tablegroup

Function Definition

Table group is a logical concept that represents a set of tables.

By default, data between different tables is randomly distributed and has no relationship. By defining a table group, you can control the proximity of a group of tables in physical storage.

Tablegroup

Reason for Introduction

OceanBase is a native distributed database. When there are multiple observer nodes in each zone in the user's deployment environment, in the same zone, data in different tables may be distributed on different machines for load balancing, and data in different partitions of the same table may also be distributed on different machines.

This will cause two problems:

- **When the partition definitions (partition type, number of partitions, partition value, etc.) of two partition tables are exactly the same, the corresponding same partitions in the two tables may be on two different nodes. Each time a JOIN operation is performed on the partition keys of the two tables, cross-machine communication is required for the data in the same partition, which may result in a large network overhead.**

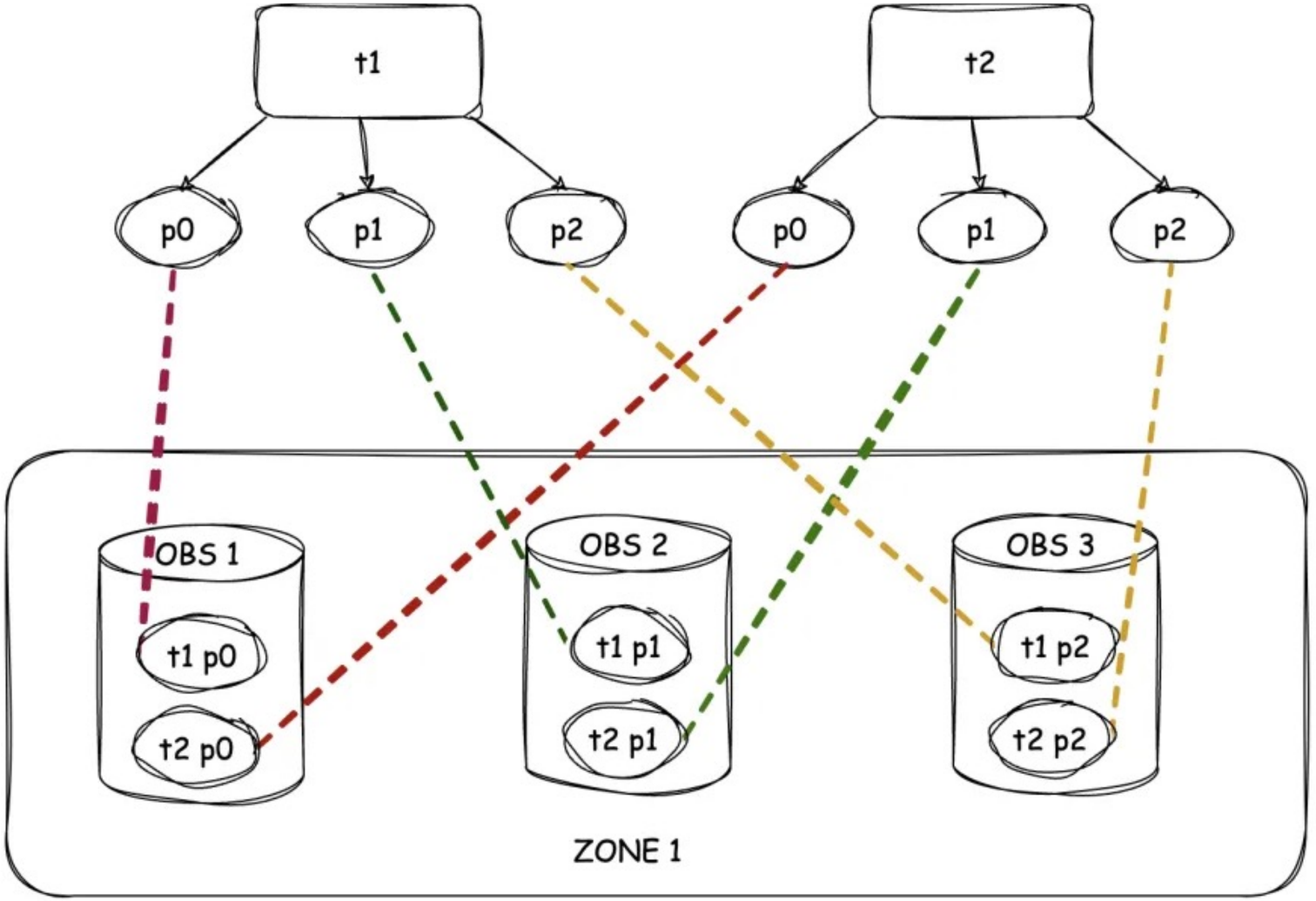
- Similarly, when tables (or partitions) that often appear in the same transaction cannot be distributed on the same node, distributed transactions will occur, which may affect transaction performance.

To improve database performance, the above cross-machine actions should be avoided as much as possible. OceanBase can use the table group function to aggregate a batch of tables on the same machine or aggregate the same partitions in tables with the same partition attributes on the same machine.

Tablegroup

Reason for Introduction

After adding tablegroup, the data of the same partition of tables t1 and t2 will be distributed to the same node, as shown in the following figure:



```
-- Create two tables t1 and t2 with the same partition rules
obclient [test]> CREATE TABLE t1(col1 int, col2 int)
                 PARTITION BY RANGE(col1)(
                   PARTITION p0 VALUES LESS THAN(100),
                   PARTITION p1 VALUES LESS THAN(200),
                   PARTITION p2 VALUES LESS THAN(300));
Query OK, 0 rows affected (0.206 sec)

obclient [test]> CREATE TABLE t2(col1 int, col2 int)
                 PARTITION BY RANGE(col1)(
                   PARTITION p0 VALUES LESS THAN(100),
                   PARTITION p1 VALUES LESS THAN(200),
                   PARTITION p2 VALUES LESS THAN(300));
Query OK, 0 rows affected (0.209 sec)
```

```
-- Create a tablegroup and add t1 and t2 to this tablegroup
obclient [test]> CREATE TABLEGROUP tg1 sharding = 'ADAPTIVE';
Query OK, 0 rows affected (0.101 sec)

obclient [test]> ALTER TABLEGROUP tg1 add t1, t2;
Query OK, 0 rows affected (0.144 sec)

obclient [test]> SHOW TABLEGROUPS WHERE tablegroup_name = 'tg1';
+-----------------+------------+---------------+----------+
| Tablegroup_name | Table_name | Database_name | Sharding |
+-----------------+------------+---------------+----------+
| tg1             | t1         | test          | ADAPTIVE |
| tg1             | t2         | test          | ADAPTIVE |
+-----------------+------------+---------------+----------+
2 rows in set (0.038 sec)
```

Tablegroup

-- Create a table with different partition rules

```
obclient [test]> CREATE TABLE t3(col1 int, col2 int)
                 PARTITION BY RANGE(col1)(
                   PARTITION p0 VALUES LESS THAN(400),
                   PARTITION p1 VALUES LESS THAN(500),
                   PARTITION p2 VALUES LESS THAN(600));
Query OK, 0 rows affected (0.206 sec)
```

-- When you try to add a table with different partitioning rules to TABLEGROUP, it will fail with an error.

```
obclient [test]> ALTER TABLEGROUP tg1 add t3;
ERROR 4179 (HY000): range_part partition value not equal, add table to tablegroup not allowed
```

Join Operator

```
explain select * from t1, t3 where t1.col1 = t3.col1;
+-----------------------------------------------------------------------------------------------+
| Query Plan                                                                                    |
+-----------------------------------------------------------------------------------------------+
| ============================================================                                  |
| |ID|OPERATOR                         |NAME    |EST.ROWS|EST.TIME(us)|                      |
| ------------------------------------------------------------                                  |
| |0 |PX COORDINATOR                   |        |1       |27          |                      |
| |1 |└─EXCHANGE OUT DISTR             |:EX10001|1       |26          |                      |
| |2 |  └─HASH JOIN                    |        |1       |25          |                      |
| |3 |    ├─EXCHANGE IN DISTR          |        |1       |13          |                      |
| |4 |    │ └─EXCHANGE OUT DISTR (PKEY)|:EX10000|1       |13          |                      |
| |5 |    │   └─PX PARTITION ITERATOR  |        |1       |12          |                      |
| |6 |    │     └─TABLE FULL SCAN      |t1      |1       |12          |                      |
| |7 |    └─PX PARTITION ITERATOR      |        |1       |12          |                      |
| |8 |      └─TABLE FULL SCAN          |t3      |1       |12          |                      |
| ============================================================                                  |
| Outputs & filters:                                                                            |
| -------------------------------------                                                         |
|   0 - output([INTERNAL_FUNCTION(t1.col1, t1.col2, t3.col1, t3.col2)]), filter(nil), rowset=16 |
|   1 - output([INTERNAL_FUNCTION(t1.col1, t1.col2, t3.col1, t3.col2)]), filter(nil), rowset=16 |
|       dop=1                                                                                   |
|   2 - output([t1.col1], [t3.col1], [t1.col2], [t3.col2]), filter(nil), rowset=16              |
|       equal_conds([t1.col1 = t3.col1]), other_conds(nil)                                      |
|   3 - output([t1.col1], [t1.col2]), filter(nil), rowset=16                                    |
|   4 - output([t1.col1], [t1.col2]), filter(nil), rowset=16                                    |
|       (#keys=1, [t1.col1]), dop=1                                                             |
|   5 - output([t1.col1], [t1.col2]), filter(nil), rowset=16                                    |
|       force partition granule                                                                 |
|   6 - output([t1.col1], [t1.col2]), filter(nil), rowset=16                                    |
|       access([t1.col1], [t1.col2]), partitions(p[0-2])                                        |
|       is_index_back=false, is_global_index=false,                                             |
|       range_key([t1.__pk_increment]), range(MIN ; MAX)always true                             |
|   7 - output([t3.col1], [t3.col2]), filter(nil), rowset=16                                    |
|       affinitize, force partition granule                                                     |
|   8 - output([t3.col1], [t3.col2]), filter(nil), rowset=16                                    |
|       access([t3.col1], [t3.col2]), partitions(p[0-2])                                        |
|       is_index_back=false, is_global_index=false,                                             |
|       range_key([t3.__pk_increment]), range(MIN ; MAX)always true                             |
+-----------------------------------------------------------------------------------------------+
35 rows in set (0.138 sec)
```

-- This partitioning rule is exactly the same, and the corresponding partitions are also distributed on the same nodes.
-- When performing join calculations on partition keys, there is no need to redistribute data in each partition, and no need to go through the network.
-- OceanBase internally calls this special join a partition-wise join, which can save a lot of network transmission overhead.

```
obclient [test]> explain select * from t1, t2 where t1.col1 = t2.col1;
+----------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                               |
+----------------------------------------------------------------------------------------------------------+
| ============================================================                                             |
| |ID|OPERATOR                  |NAME    |EST.ROWS|EST.TIME(us)|                                          |
| ------------------------------------------------------------                                             |
| |0 |PX COORDINATOR            |        |1       |26          |                                          |
| |1 |└─EXCHANGE OUT DISTR      |:EX10000|1       |25          |                                          |
| |2 |  └─PX PARTITION ITERATOR |        |1       |24          |                                          |
| |3 |    └─HASH JOIN           |        |1       |24          |                                          |
| |4 |      ├─TABLE FULL SCAN   |t1      |1       |12          |                                          |
| |5 |      └─TABLE FULL SCAN   |t2      |1       |12          |                                          |
| ============================================================                                             |
| Outputs & filters:                                                                                       |
| -------------------------------------                                                                    |
|   0 - output([INTERNAL_FUNCTION(t1.col1, t1.col2, t2.col1, t2.col2)]), filter(nil), rowset=16            |
|   1 - output([INTERNAL_FUNCTION(t1.col1, t1.col2, t2.col1, t2.col2)]), filter(nil), rowset=16            |
|       dop=1                                                                                              |
|   2 - output([t1.col1], [t2.col1], [t1.col2], [t2.col2]), filter(nil), rowset=16                         |
|       partition wise, force partition granule                                                            |
|   3 - output([t1.col1], [t2.col1], [t1.col2], [t2.col2]), filter(nil), rowset=16                         |
|       equal_conds([t1.col1 = t2.col1]), other_conds(nil)                                                 |
|   4 - output([t1.col1], [t1.col2]), filter(nil), rowset=16                                               |
|       access([t1.col1], [t1.col2]), partitions(p[0-2])                                                   |
|       is_index_back=false, is_global_index=false,                                                        |
|       range_key([t1.__pk_increment]), range(MIN ; MAX)always true                                        |
|   5 - output([t2.col1], [t2.col2]), filter(nil), rowset=16                                               |
|       access([t2.col1], [t2.col2]), partitions(p[0-2])                                                   |
|       is_index_back=false, is_global_index=false,                                                        |
|       range_key([t2.__pk_increment]), range(MIN ; MAX)always true                                        |
+----------------------------------------------------------------------------------------------------------+
27 rows in set (0.230 sec)
```

Tablegroup

Reason for Introduction

OceanBase is a native distributed database. When there are multiple observer nodes in each zone in the user's deployment environment, in the same zone, data in different tables may be distributed on different machines for load balancing. Data in different partitions of the same table may also be distributed on different machines.

This will cause two problems:

- When the partition definitions (partition type, number of partitions, partition value, etc.) of two partition tables are the same, the corresponding partitions in the two tables may be on two different nodes. Each time a JOIN operation is performed on the partition keys of these two tables, cross-machine communication is required for the data of the same partition, which may incur a large network overhead.

- Similarly, when tables (or partitions) that often appear in the same transaction cannot be distributed on the same node, distributed transactions will occur, which may affect transaction performance.

To improve database performance, the above cross-machine actions should be avoided as much as possible. OceanBase can use the table group function to aggregate a batch of tables on the same machine or aggregate the same partitions in tables with the same partition attributes on the same machine.

Tablegroup

Related Syntax

For details on the properties and syntax of table groups, see the "Creating and Managing Table Groups" section in the OceanBase official website, which will not be repeated here.

```
-- Create a tablegroup and add t1 and t2 to this tablegroup
obclient [test]> CREATE TABLEGROUP tg1 sharding = 'ADAPTIVE';
Query OK, 0 rows affected (0.101 sec)

obclient [test]> ALTER TABLEGROUP tg1 add t1, t2;
Query OK, 0 rows affected (0.144 sec)

obclient [test]> SHOW TABLEGROUPS WHERE tablegroup_name = 'tg1';
+-----------------+------------+---------------+----------+
| Tablegroup_name | Table_name | Database_name | Sharding |
+-----------------+------------+---------------+----------+
| tg1             | t1         | test          | ADAPTIVE |
| tg1             | t2         | test          | ADAPTIVE |
+-----------------+------------+---------------+----------+
2 rows in set (0.038 sec)
```

Tablegroup

Sharding Properties

OceanBase has introduced the concept and capabilities of Table Group since V1.X.

Multiple tables with related relationships often follow the same partitioning rules. By clustering and distributing partitions with the same rules together, Partition-Wise Join can be implemented, greatly optimizing read and write performance. The introduction of Table Group is to transform ordinary Join operations into Partition-Wise Join.

OceanBase V4.2 introduces the SHARDING attribute for Table Groups, enabling finer control over the aggregation and dispersion of table data within a Table Group. By default, data across different tables is distributed randomly. To ensure consistent data distribution for related tables, you need to define alignment rules between their partitions. This allows partitions of related tables to be co-located on the same server, enabling Partition-Wise Joins and significantly enhancing performance. Using a Table Group with SHARDING enabled, rather than set to NONE, can fulfill this requirement.

- When SHARDING = NONE

  Table Group describes that all partitions of all tables are gathered on the same server, and does not limit the partition type of the table. This property can be used to distribute tables that often appear in the same transaction on the same node, thereby simplifying distributed transactions into single-node transactions.

- When SHARDING != NONE

  The data of each table in the Table Group is scattered and distributed on multiple machines. To ensure the same data distribution of all tables, the Table Group requires that all tables have the same partitioning method, including partition type, number of partitions, and partition value. The system will schedule the partitions with the same partition attributes to be clustered (aligned) and distributed on the same server, thereby achieving Partition-Wise Join.

Tablegroup

Sharding Properties

The following describes the meaning of different SHARDING values and their impact on the tables in the Table Group.

PARTITION: Sharding by first-level partition. If it is a second-level partition table, all second-level partitions under the first-level partition are gathered together.

- Partitioning method requirements: The partitioning methods of the first-level partitions are the same. If it is a second-level partition table, only the partitioning method of the first-level partition is verified. Therefore, the first-level partition table and the second-level partition table can exist at the same time, as long as their first-level partitions have the same partitioning method.
- Partition alignment rule: Partitions with the same first-level partition value are grouped, including the first-level partition of the first-level partition table and all second-level partitions under the corresponding first-level partition of the second-level partition table.

ADAPTIVE: Adaptive sharding mode. If the table group contains a first-level partition table, the table is sharding according to the first-level partition; if the table group contains a second-level partition table, the second-level partitions under each first-level partition are sharding.

- Partitioning method requirements: Either all are first-level partition tables or all are second-level partition tables. If it is a first-level partition table, the first-level partitions must have the same partitioning method; if it is a second-level partition table, both the first-level and second-level partitions must have the same partitioning method.
- Partition alignment rules: For a first-level partition table, partitions with the same first-level partition value are grouped; for a second-level partition table, partitions with the same first-level partition value and the same second-level partition value are grouped.

Tablegroup

Sharding Properties

Example 1: SHARDING = NONE

In a table group with SHARDING = NONE, all partitions of the table, regardless of the partitioning method, are distributed in the same partition group (partition group is a physical concept, which can be simply understood as the data of the same partition group must be distributed on the same machine)

```
SQL> CREATE TABLEGROUP TG1 SHARDING = 'NONE';

# Non-partitioned table
SQL> CREATE TABLE T_NONPART (pk int primary key) tablegroup = TG1;

# First-level partition table
SQL> CREATE TABLE T_PART_2 (pk int primary key)  tablegroup = TG1
     partition by hash(pk)
     partitions 2;

# Secondary partition table
SQL> CREATE TABLE T_SUBPART_2_2 (pk int, c1 int, primary key(pk, c1)) tablegroup = TG1
     partition by hash(pk)
     subpartition by hash(c1) subpartitions 2
     partitions 2;
```

Table	Partition Group
	0
T_NONPART	P0
T_PART_2	P0, P1
T_SUBPART_2_2	P0SP0, P0SP1, P1SP0, P1SP1

Tablegroup

Sharding Properties

Example 2: SHARDING = PARTITION

All tables in the Table Group with SHARDING = PARTITION are considered first-level partition tables. The first-level partitioning method of all tables is required to be the same, and partitions with the same first-level partition attributes are aggregated into one Partition Group.

```
SQL> CREATE TABLEGROUP TG1 SHARDING = 'PARTITION';

# First-level partition table
SQL> CREATE TABLE T_PART_2 (pk int primary key) tablegroup = TG1
     partition by hash(pk) partitions 2;

# Secondary partition table
SQL> CREATE TABLE T_SUBPART_2_2 (pk int, c1 int, primary key(pk, c1)) tablegroup = TG1
     partition by hash(pk)
     subpartition by hash(c1) subpartitions 2
     partitions 2;
```

Table	Partition Group	
	0	1
T_PART_2	P0	P1
T_SUBPART_2_2	P0SP0, P0SP1	P1SP0, P1SP1

Tablegroup

Sharding Properties

Example: SHARDING = ADAPTIVE

Table Group requires that the primary and secondary partitioning methods of all tables are completely consistent. Primary partition tables and secondary partition tables cannot be in the same Table Group.

```
SQL> CREATE TABLEGROUP TG_PART SHARDING = 'ADAPTIVE';

# First-level partition table
SQL> CREATE TABLE T1_PART_2 (pk int primary key) tablegroup = TG_PART
     partition by hash(pk) partitions 2;

# First-level partition table
SQL> CREATE TABLE T2_PART_2 (pk int primary key, c1 int) tablegroup = TG_PART
     partition by hash(pk) partitions 2;
```

Table	Partition Group	
	0	1
T1_PART_2	P0	P1
T2_PART_2	P0	P1

Tablegroup

Sharding Properties

Example: SHARDING = ADAPTIVE

Table Group requires that the primary and secondary partitioning methods of all tables are completely consistent. Primary partition tables and secondary partition tables are not supported in the same Table Group.

```
SQL> CREATE TABLEGROUP TG_SUBPART SHARDING = 'ADAPTIVE';
```

Secondary partition table

```
SQL> CREATE TABLE T1_SUBPART_2_2 (pk int, c1 int, primary key(pk, c1)) tablegroup = TG_SUBPART
     partition by hash(pk)
     subpartition by hash(c1) subpartitions 2
     partitions 2;
```

Secondary partition table

```
SQL> CREATE TABLE T2_SUBPART_2_2 (pk int, c1 int, c2 int, primary key(pk, c1)) tablegroup = TG_SUBPART
     partition by hash(pk)
     subpartition by hash(c1) subpartitions 2
     partitions 2;
```

Table	Partition Group			
	00	01	10	11
T1_SUBPART_2_2	P0SP0	P0SP1	P1SP0	P1SP1
T2_SUBPART_3_3	P0SP0	P0SP1	P1SP0	P1SP1

Tablegroup

FAQ

Is there a subordinate relationship between the table group and the database?

There is no subordinate relationship between tablegroup and databases. They are all objects under the tenant.

The subordinate relationships of common database objects are as follows:

- Tenant
  - Database / tablegroup
    - Table
      - ❖ Index / partition / constraint

Agenda

- Global Index
- Recycle Bin
- Tablegroup
- Sequence
- Self-study Content

Sequence

Function Definition

In the OceanBase database, a sequence is a unique and usually increasing value generated by the database according to certain rules. It is usually used to generate unique identifiers.

Sequence

Reason for Introduction

OceanBase has many users whose businesses originally ran on DB2/Oracle, but they are planning to choose the MySQL technology route in the future, so they need to migrate tenants from DB2/Oracle to Oceanbase MySQL mode.

To reduce the complexity of transforming businesses that previously used sequences extensively in DB2/Oracle, OceanBase supports sequence functions that are compatible with Oracle behavior in MySQL mode.

Sequence

Related Syntax

For details, see the "Creating and Managing Sequences" section in the OceanBase official website. The syntax is compatible with Oracle and will not be repeated in this document.

Sequence

Applicable Scenarios

1. The scenario of migrating tenants from DB2/Oracle to Oceanbase MySQL mode.

2. The feature of binding the increment column to the table cannot meet the business requirements. Sequences are not bound to tables and can be created independently and used across tables.

3. The increment column does not have the CYCLE capability and will stop working after reaching MAXVALUE, which cannot meet the business requirements. Sequences support cyclic sequences and have CYCLE capabilities.

Sequence

FAQ

1. What are the similarities and differences between a sequence and an increment column?

   1. The increment column is bound to the table. The sequence is not bound to the table and can be created independently and used across tables.

   2. The increment column does not have the CYCLE capability. The sequence supports cyclic sequences and has the CYCLE capability.

Sequence

FAQ

-- Create a table with an auto-increment column id, and strongly bind the auto-increment column to the table

```
obclient [test]> CREATE TABLE t1(id bigint not null auto_increment primary key, name varchar(50));
Query OK, 0 rows affected (0.489 sec)

obclient [test]> INSERT INTO t1(name) VALUES('A'),('B'),('C');
Query OK, 3 rows affected (0.036 sec)

obclient [test]> SELECT * FROM t1;
+----+------+
| id | name |
+----+------+
|  1 | A    |
|  2 | B    |
|  3 | C    |
+----+------+
3 rows in set (0.021 sec)
```

-- Create a sequence starting at 1, with a minimum value of 1, a maximum value of 5, and a step size of 2. The sequence values are generated without cycling.

```
obclient [test]> CREATE SEQUENCE seq1 START WITH 1 MINVALUE 1 MAXVALUE 5 INCREMENT BY 2 NOCYCLE;
Query OK, 0 rows affected (0.073 sec)

obclient [test]> SELECT seq1.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       1 |
+---------+
1 row in set (0.012 sec)

obclient [test]> SELECT seq1.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       3 |
+---------+
1 row in set (0.004 sec)

obclient [test]> SELECT seq1.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       5 |
+---------+
1 row in set (0.004 sec)
```

-- If NOCYCLE is set, after reaching MAXVALUE, no further sequences can be generated.

```
obclient [test]> SELECT seq1.nextval FROM DUAL;
ERROR 4332 (HY000): sequence exceeds MAXVALUE and cannot be instantiated
```

-- Create another sequence with a starting value of 1, a minimum value of 1, a maximum value of 5, and a step size of 2. The values of the sequence are generated cyclically (the number of auto-increment values pre-allocated in memory is 2)


```
obclient [test]> CREATE SEQUENCE seq7 START WITH 1 MINVALUE 1 MAXVALUE 5 INCREMENT BY 2 CYCLE CACHE 2;
Query OK, 0 rows affected (0.095 sec)

obclient [test]> SELECT seq7.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       1 |
+---------+
1 row in set (0.009 sec)

obclient [test]> SELECT seq7.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       3 |
+---------+
1 row in set (0.005 sec)

obclient [test]> SELECT seq7.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       5 |
+---------+
1 row in set (0.005 sec)

obclient [test]> SELECT seq7.nextval FROM DUAL;
+---------+
| nextval |
+---------+
|       1 |
+---------+
1 row in set (0.001 sec)
```

-- Sequences can be used in INSERT and UPDATE in addition to top-level SELECT.

```
obclient [test]> create table t2(c1 int);
Query OK, 0 rows affected (0.192 sec)

obclient [test]> insert into t2 values(seq7.nextval);
Query OK, 1 row affected (0.009 sec)

obclient [test]> select * from t2;
+------+
| c1   |
+------+
|    3 |
+------+
1 row in set (0.001 sec)

obclient [test]> update t2 set c1 = seq7.nextval;
Query OK, 1 row affected (0.010 sec)
Rows matched: 1  Changed: 1  Warnings: 0

obclient [test]> select * from t2;
+------+
| c1   |
+------+
|    5 |
+------+
1 row in set (0.001 sec)
```

Sequence

FAQ

2. What effect does the ORDER | NOORDER attribute of sequence and auto-increment columns have?

The ORDER | NOORDER attribute of sequence and auto-increment columns has the same effect. The following takes the auto-increment column as an example to illustrate.

As a distributed database, OceanBase tables are usually distributed on multiple different machines. Because it is necessary to ensure the performance of auto-increment column generation in a distributed multi-machine scenario, there will be jump problems in the auto-increment value generation process.

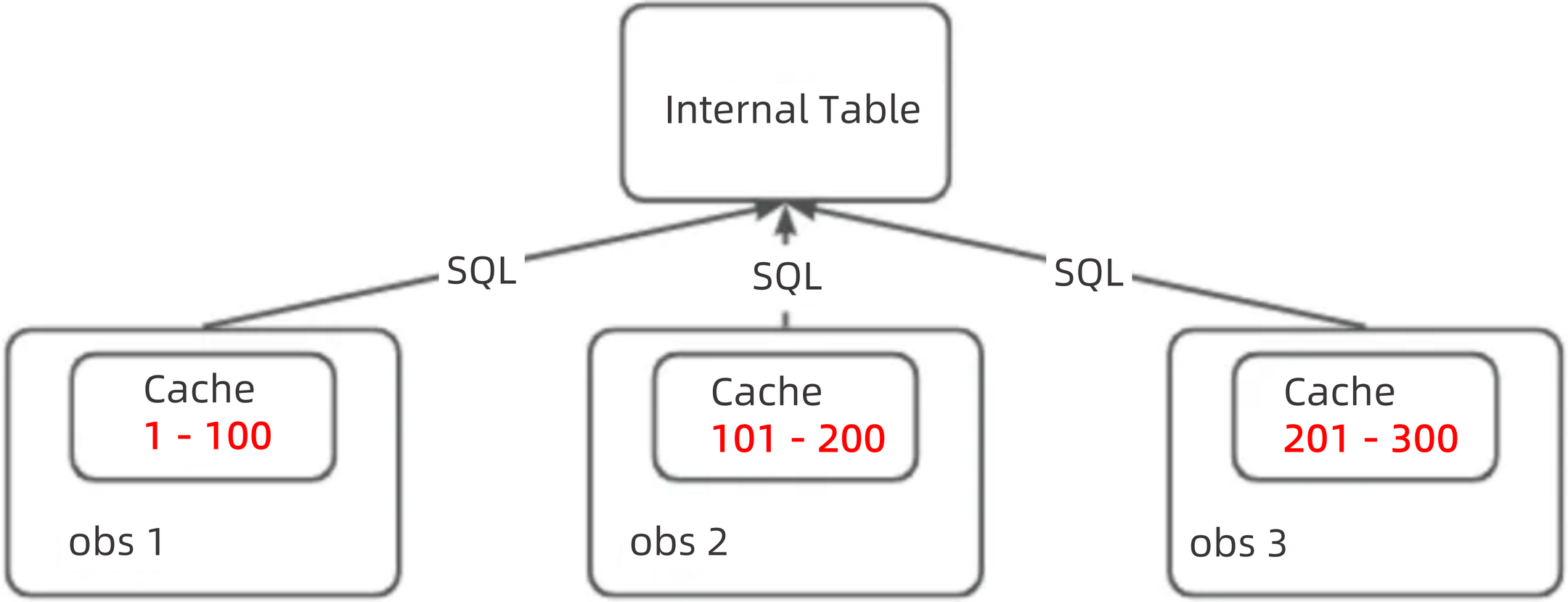
In OceanBase, auto-increment columns support two auto-increment modes, namely NOORDER mode and ORDER mode. The default mode is the ORDER mode.

Sequence

FAQ

NOORDER Mode

For auto-increment columns based on distributed cache, after setting this mode, only the global uniqueness of the value is guaranteed. The auto-increment value is only incremented locally within the node, not globally.

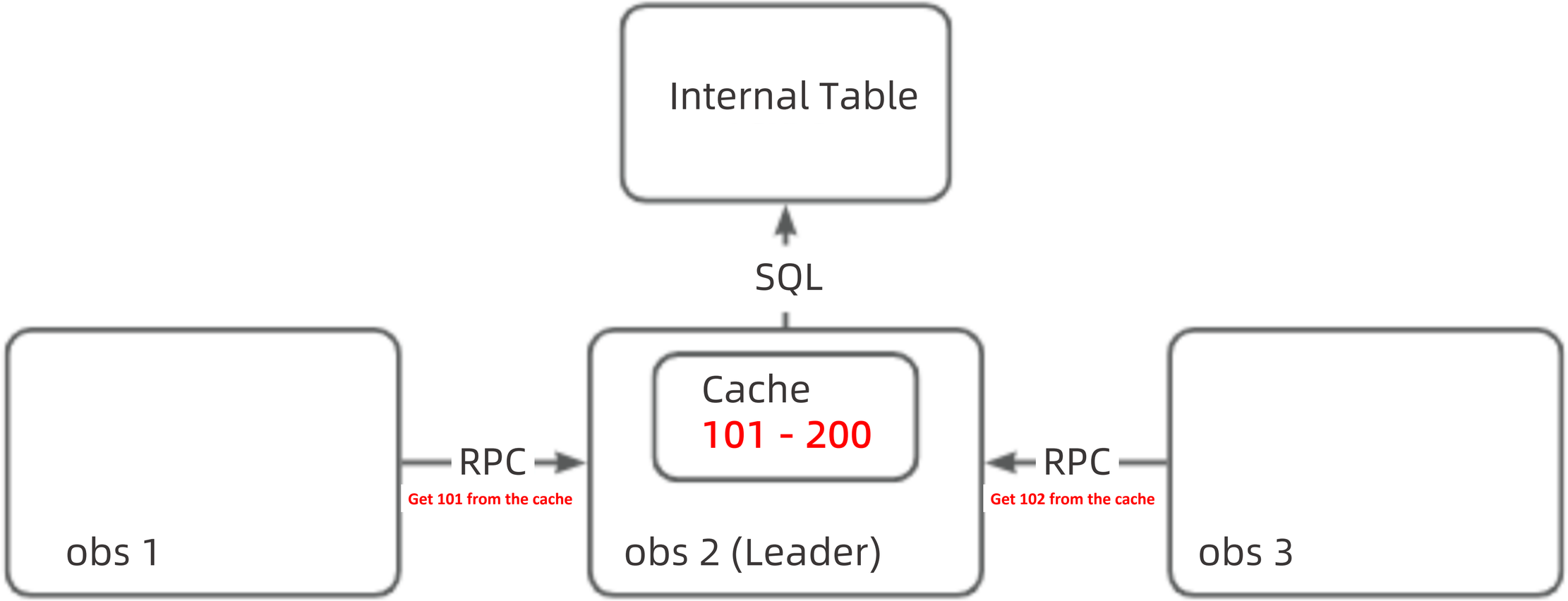


Sequence

FAQ

ORDER Mode

Auto-increment column based on the centralized cache. After setting this mode, the values of the sequence and auto-increment column are incremented globally.



Sequence

FAQ

If performance overhead is considered, how should related properties be set when creating a sequence?

When creating a sequence, if you set the ORDER attribute, in order to ensure global order, each operation to obtain NEXTVALUE needs to go to the central node to update a specific internal table. In high-concurrency scenarios, there may be a high lock conflict. If you do not require the sequence value to be incremental, but only require it to be unique, it is recommended to set the sequence attribute to NOORDER.

At the same time, when the performance requirements are high, you should also pay attention to the CACHE / NOCACHE attribute:

- NOCACHE: Indicates that the auto-increment value is not cached in OBServer. In this mode, each call to NEXTVAL will trigger an internal table SELECT and UPDATE, which will affect the performance of the database.

- CACHE: Used to specify the number of auto-increment cached in each OBServer memory. The default value is 20 (when creating a sequence, it is recommended to declare it manually because the default CACHE value is too small. When the single-machine TPS is 100, it is recommended to set CACHE SIZE to 360000).

Agenda

- Global Index
- Recycle Bin
- Tablegroup
- Sequence
- Self-study Content

Self-study Content

Flashback Query (self-study content)

OceanBase provides a record-level Flashback Query function, which allows users to obtain data from a certain historical version.
Please refer to the "Flashback Query" section on the official website to complete the self-study.

Replicated Table (self-study content)

OceanBase provides the option of replicating table properties when creating a table. Replicating table properties means that the latest changes to the data can be read on any healthy replica. For users who have a low write frequency and are more concerned about read operation latency and load balancing, replicating tables is a good choice.
Please refer to the "OceanBase v4.2 Replicating Table Feature Description" section in the community blog to complete your self-study.

OCEANBASE

FROM INTRODUCTION TO PRACTICE

Thank You!

OceanBase Official website:
https://oceanbase.github.io/

GitHub Discussions:
https://github.com/oceanbase/oceanbase/discussions