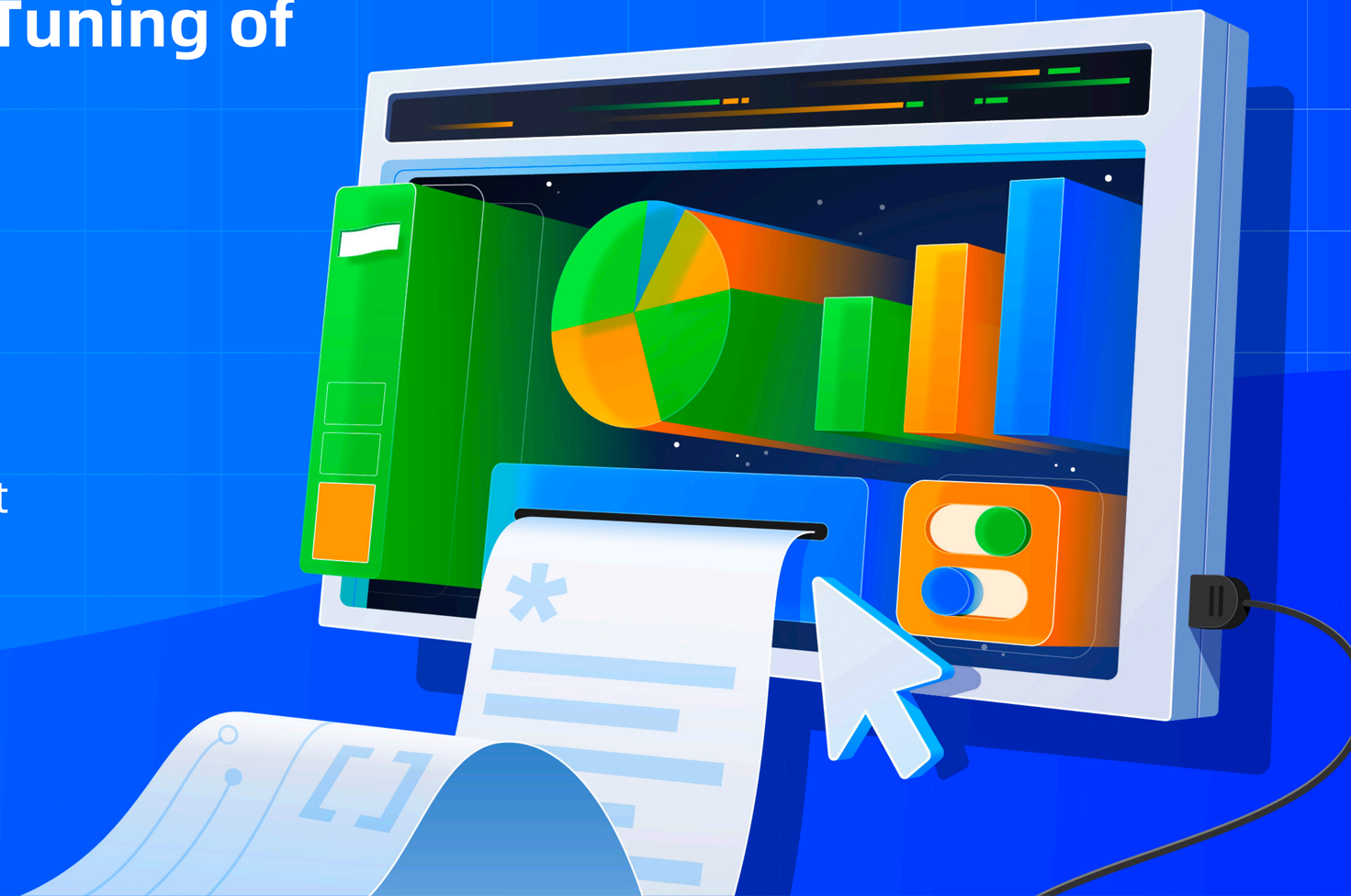
FROM INTRODUCTION TO PRACTICE

Lesson 8: Diagnosis and Tuning of OceanBase

Peng Wang

OceanBase Global Technical Evangelist



Agenda

Lesson 8.1

- ODP SQL Routing Principles
- Managing Database Connections
- Analyzing SQL Monitoring Views
- Read and Manage SQL Execution Plans

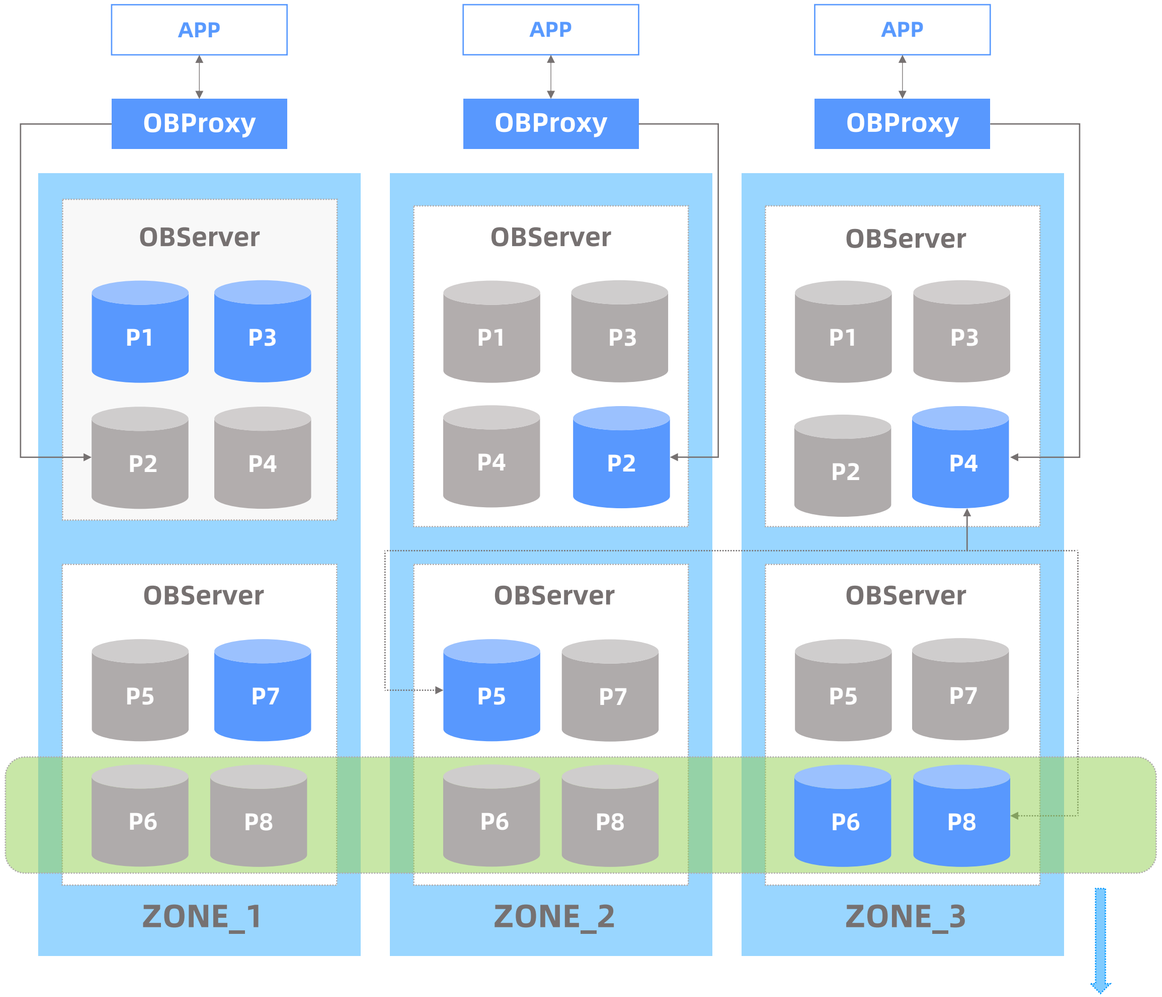
Lesson 8.2

- Common SQL Tuning Methods
- Troubleshooting Ideas for SQL Performance Issues

OceanBase Distributed Architecture

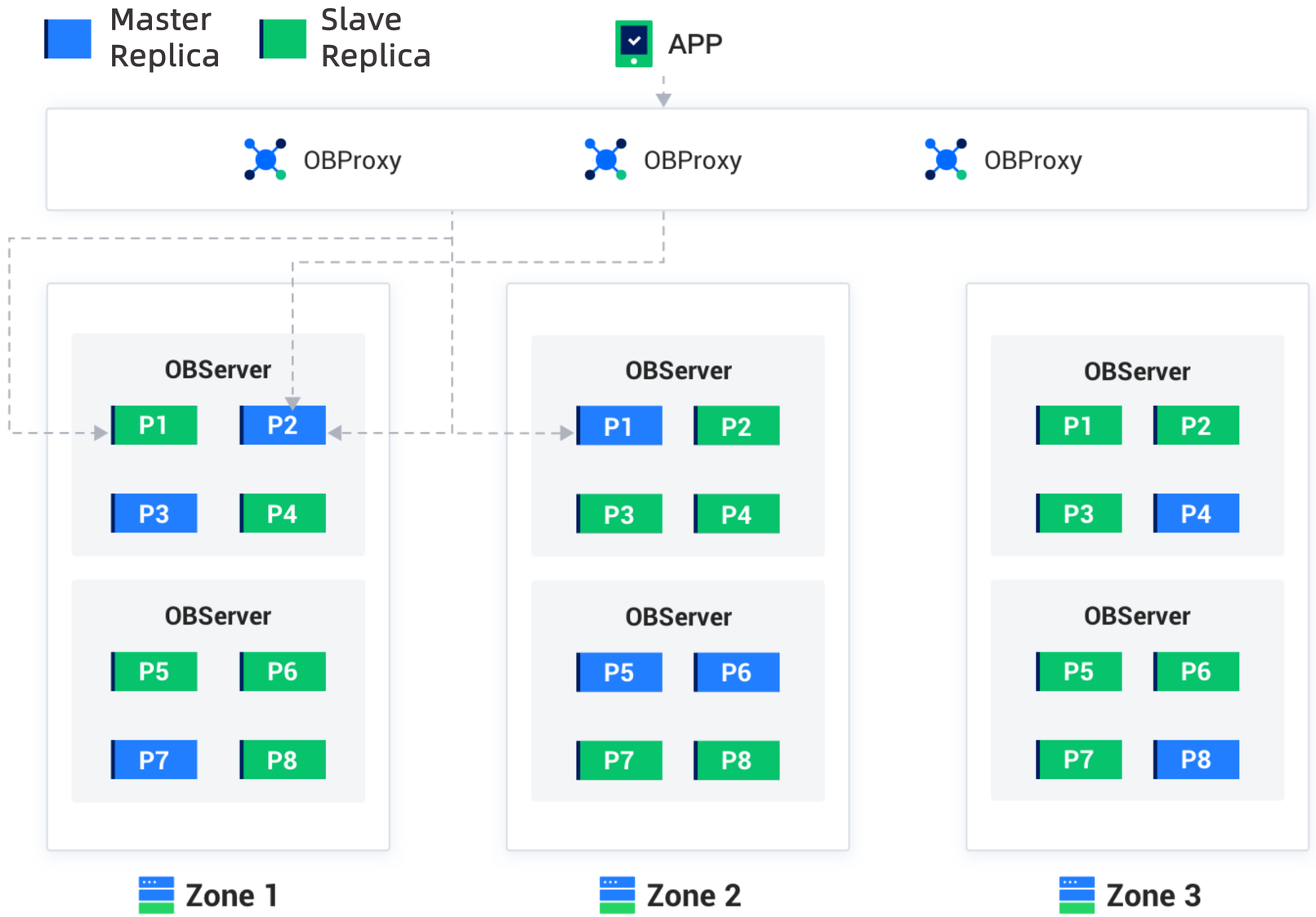
Paxos Protocol + Shared-nothing Architecture + Partition-level High Availability

- **Multiple replicas:** Generally deployed as three or five zones, each zone consists of multiple server nodes (OBServer)
- **Peer nodes:** Each node has its own SQL engine and storage engine, independently manages the data partitions it carries, TCP/IP intercommunication, and collaborative services
- **No need for storage device sharing:** Data is distributed on each node, not based on any device-level shared storage technology, and does not require a SAN network
- **Partition-level availability:** Partitions are the basic unit of reliability and scalability, automatically implementing access routing, policy-driven load balancing, and autonomous fault recovery
- **High availability + strong consistency:** efficient and reliable engineering implementation of multiple replicas + Paxos distributed protocol, ensuring data (log) persistence is successful on the majority of nodes

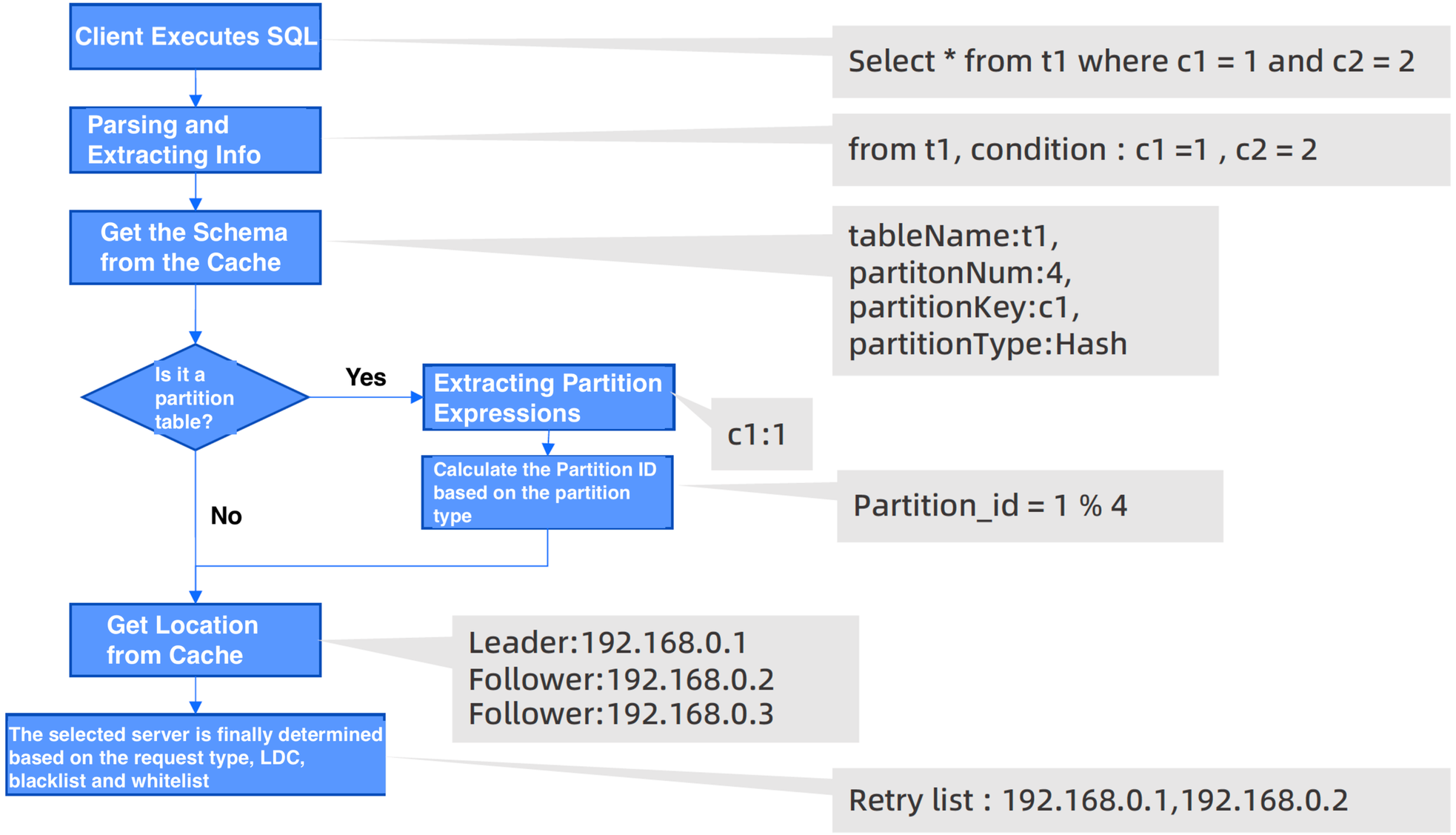


OceanBase Three-replica architecture

Functions of ODP

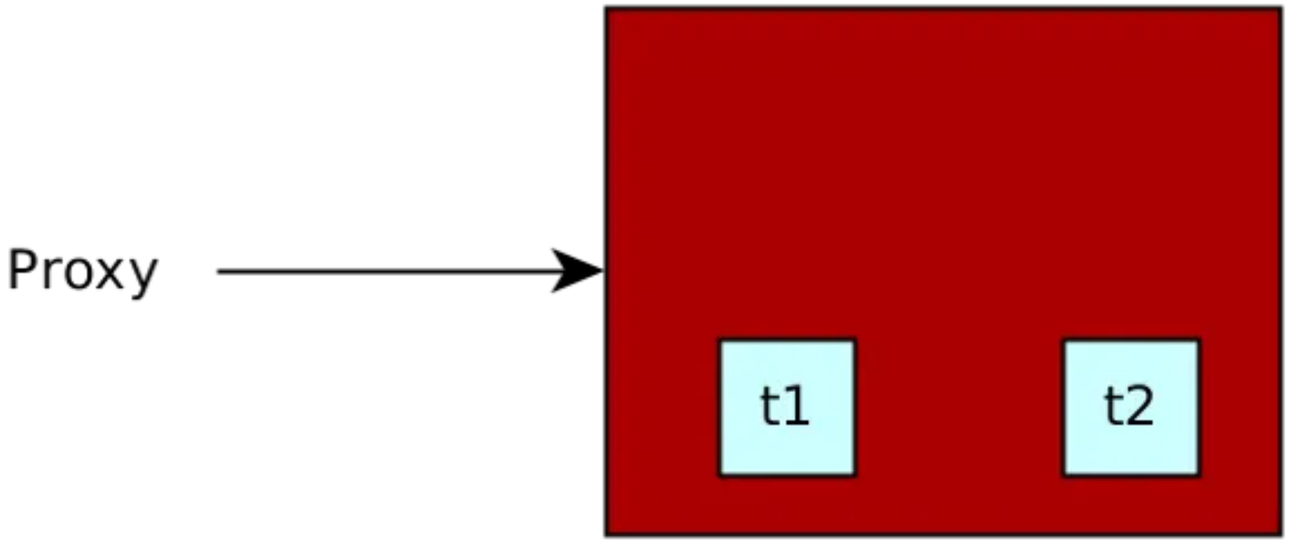


Implementation Logic of ODP Routing



SQL Plan Type

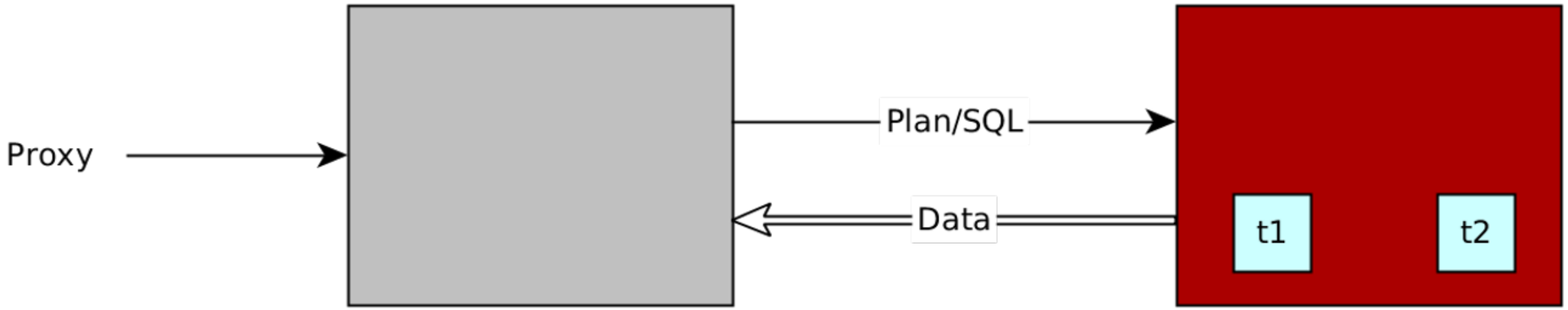
Local Plan




```
=======================================
|ID|OPERATOR    |NAME|EST. ROWS|COST     |
---------------------------------------
|0 |HASH JOIN   |    |98010000 |66774608|
|1 |  TABLE SCAN|T1  |100000   |68478    |
|2 |  TABLE SCAN|T2  |100000   |68478    |
=======================================
```

SQL Plan Type

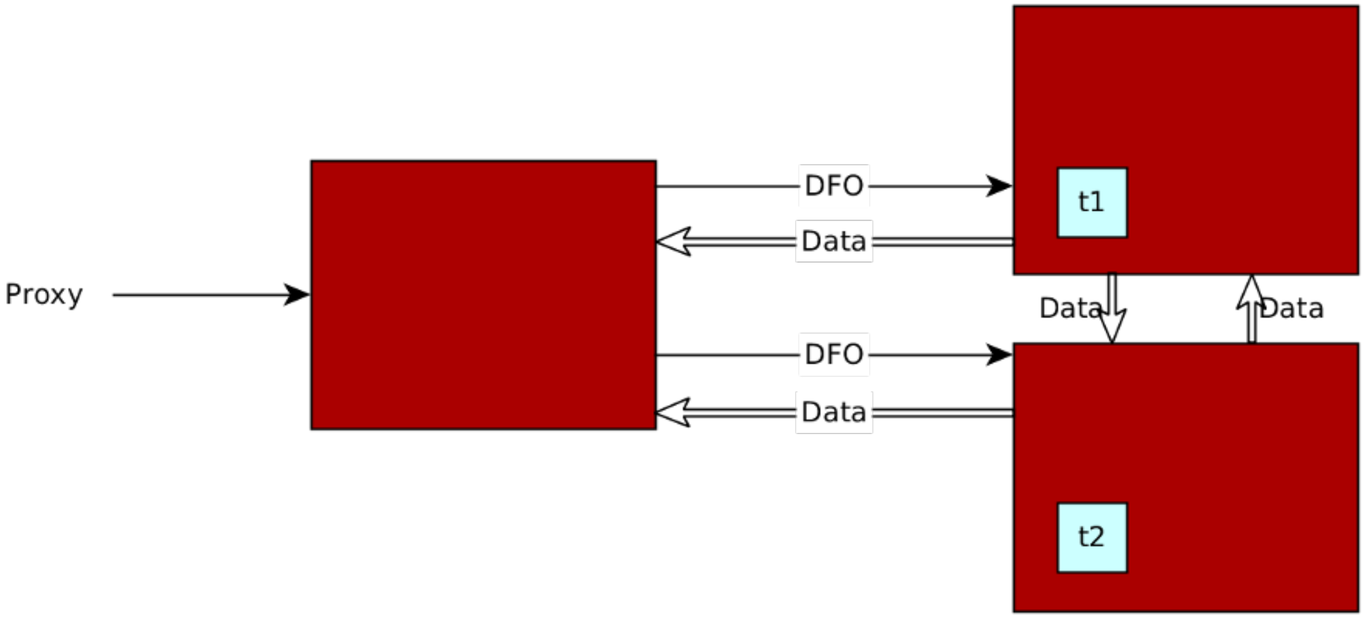
Long-range Planning




```
=============================================
|ID|OPERATOR              |NAME|EST. ROWS|COST     |
---------------------------------------------
|0 |EXCHANGE IN REMOTE    |    |98010000 |154912123|
|1 | EXCHANGE OUT REMOTE  |    |98010000 |66774608 |
|2 |  HASH JOIN           |    |98010000 |66774608 |
|3 |   TABLE SCAN         |T1  |100000   |68478    |
|4 |   TABLE SCAN         |T2  |100000   |68478    |
=============================================
```

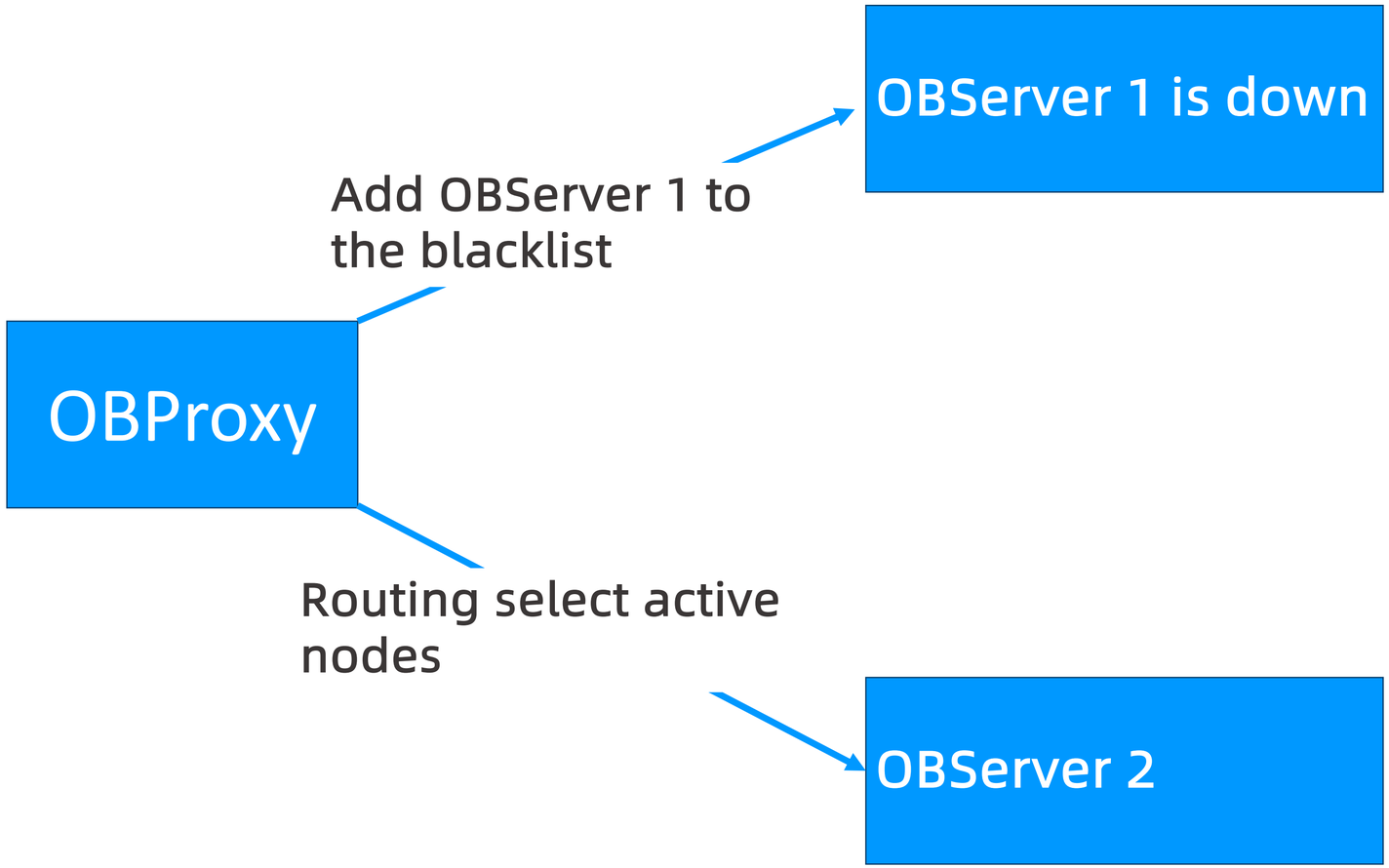
SQL Plan Type

Distributed Planning



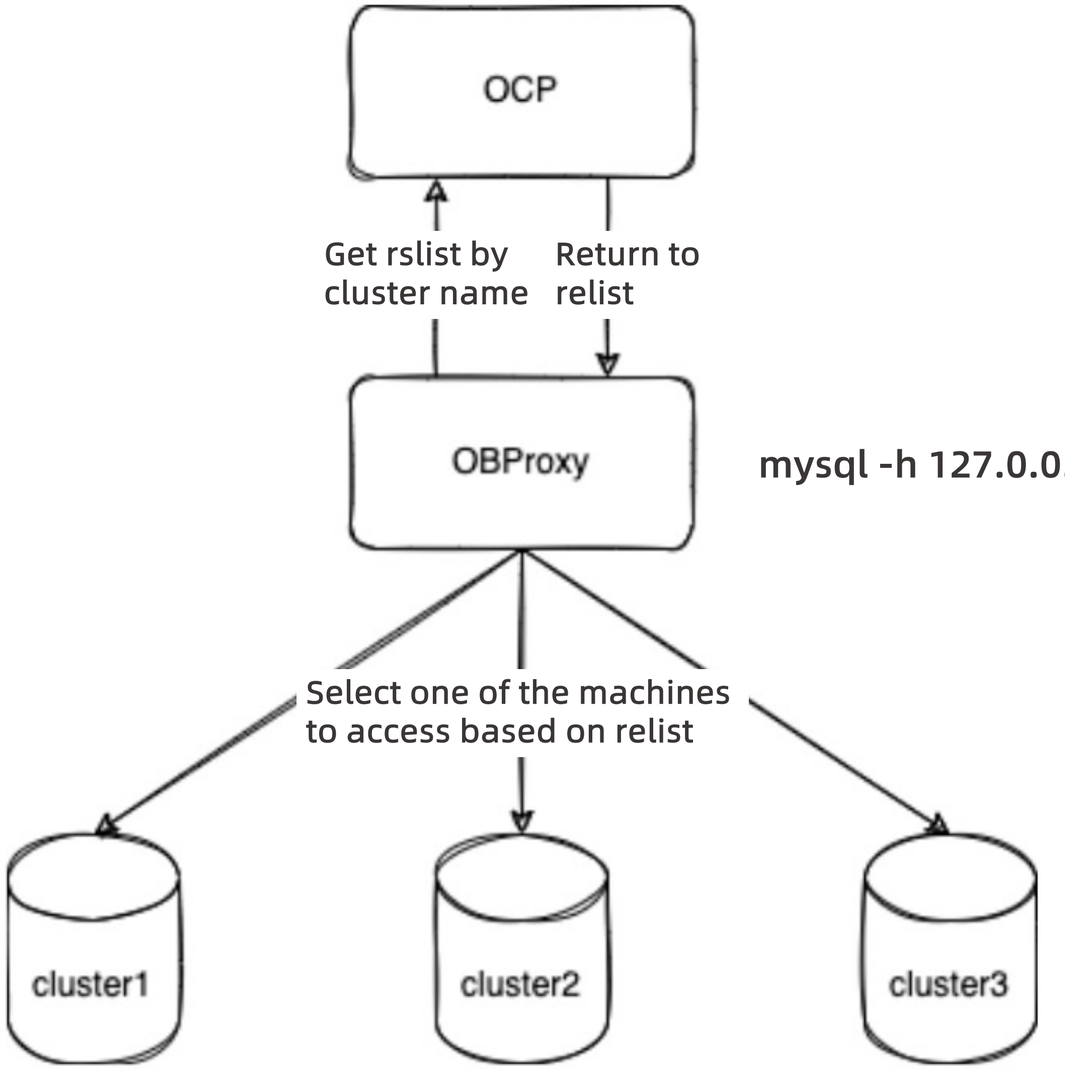

```
==========================================================
|ID|OPERATOR                      |NAME     |EST. ROWS|COST      |
----------------------------------------------------------
|0 |PX COORDINATOR                |         |980100000|1546175452|
|1 | EXCHANGE OUT DISTR           |:EX10002|980100000|664800304 |
|2 |  HASH JOIN                   |         |980100000|664800304 |
|3 |   EXCHANGE IN DISTR          |         |200000   |213647    |
|4 |    EXCHANGE OUT DISTR (HASH)|:EX10000|200000   |123720    |
|5 |     PX BLOCK ITERATOR        |         |200000   |123720    |
|6 |      TABLE SCAN              |T1       |200000   |123720    |
|7 |   EXCHANGE IN DISTR          |         |500000   |534080    |
|8 |    EXCHANGE OUT DISTR (HASH)|:EX10001|500000   |309262    |
|9 |     PX BLOCK ITERATOR        |         |500000   |309262    |
|10|      TABLE SCAN              |T2       |500000   |309262    |
==========================================================
```

High Availability Factors of ODP Routing



ODP Routing Features and Strategies

Cluster Routing



mysql -h 127.0.0.1 -u user_name@tenant_name#cluster_name -P2883

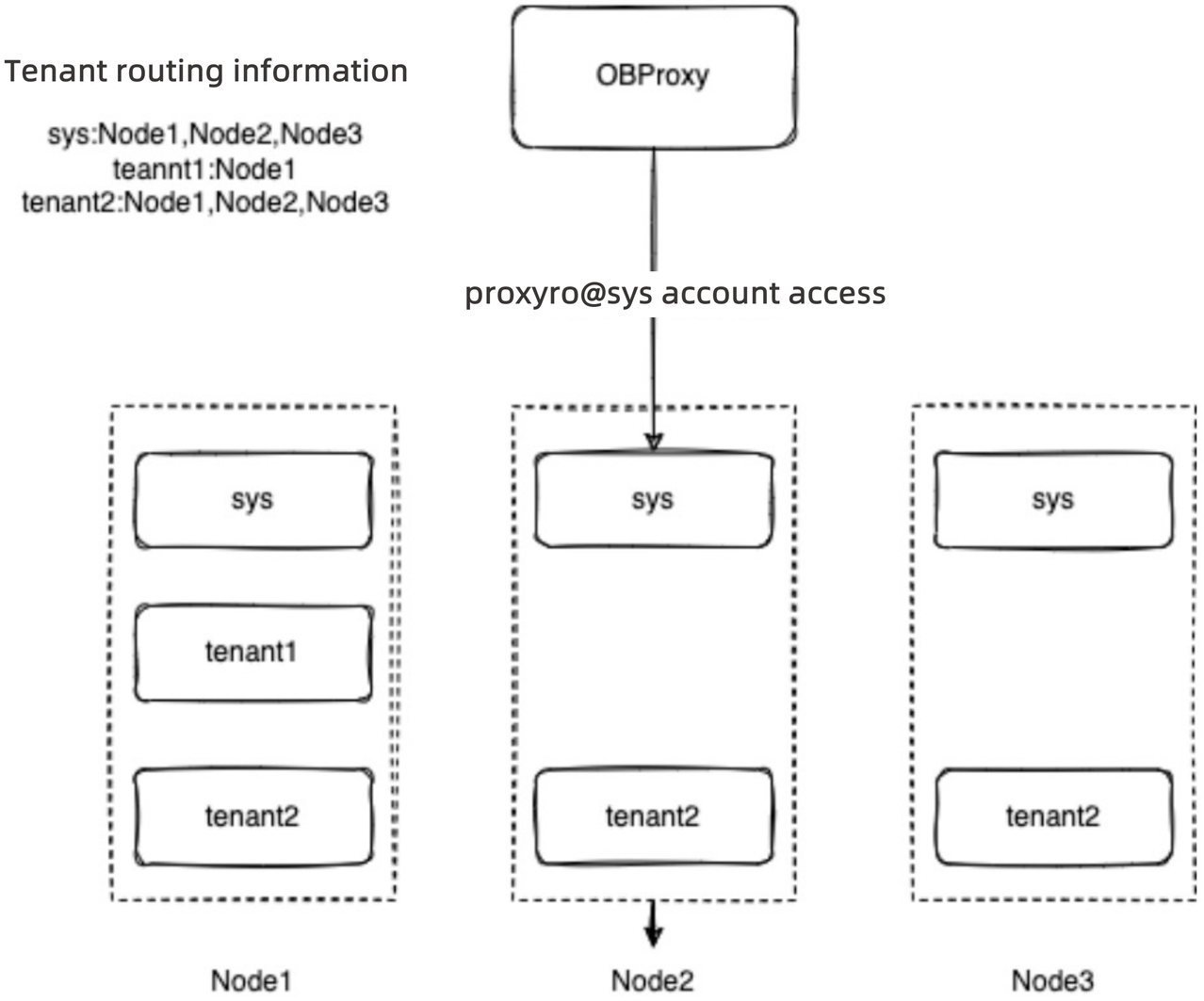
mysql -h 1.2.3.4 -u user_name@tenant_name -P12345

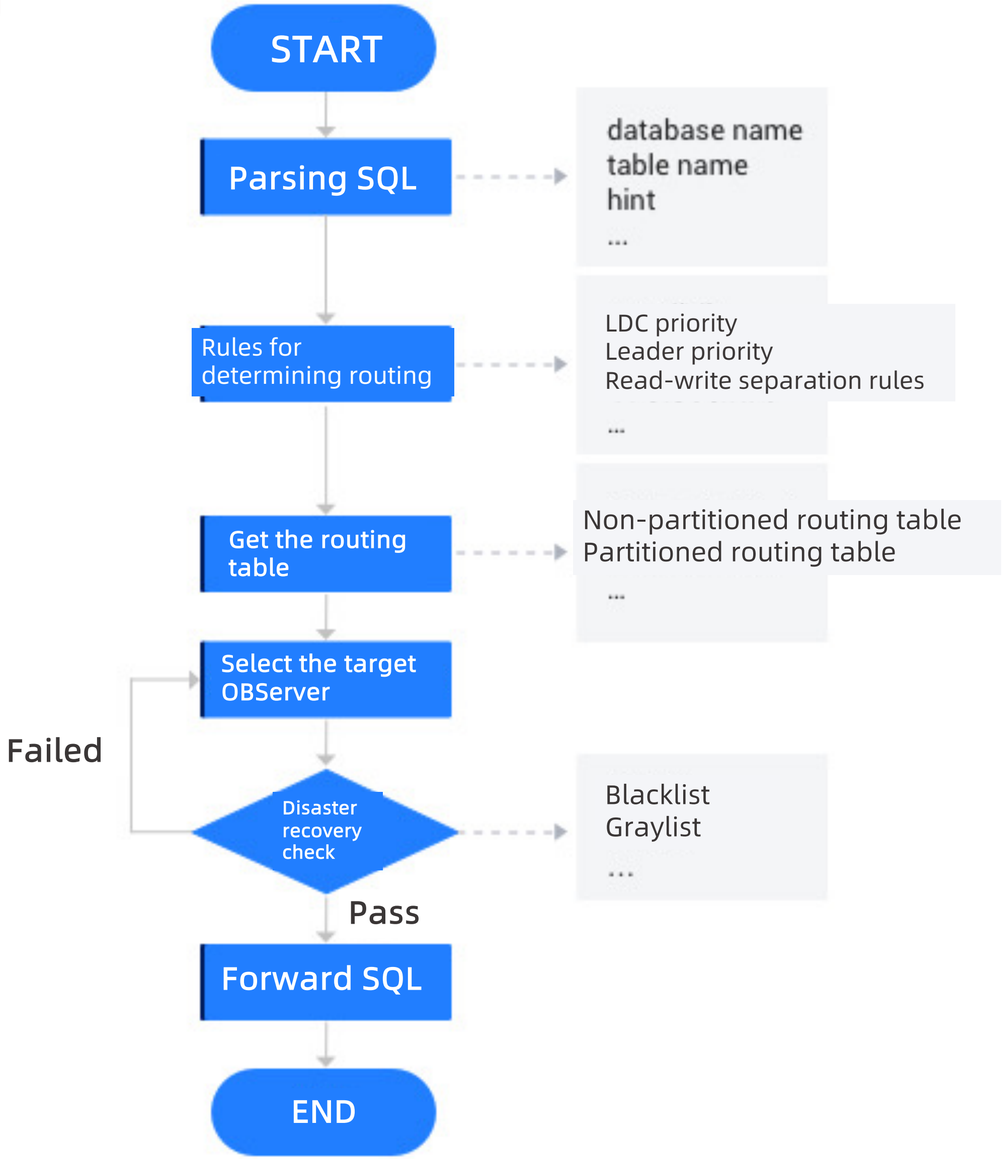
ODP Routing Features and Strategies

Tenant Routing

Tenant routing information

sys:Node1,Node2,Node3
teannt1:Node1
tenant2:Node1,Node2,Node3





Agenda

Lesson 8.1

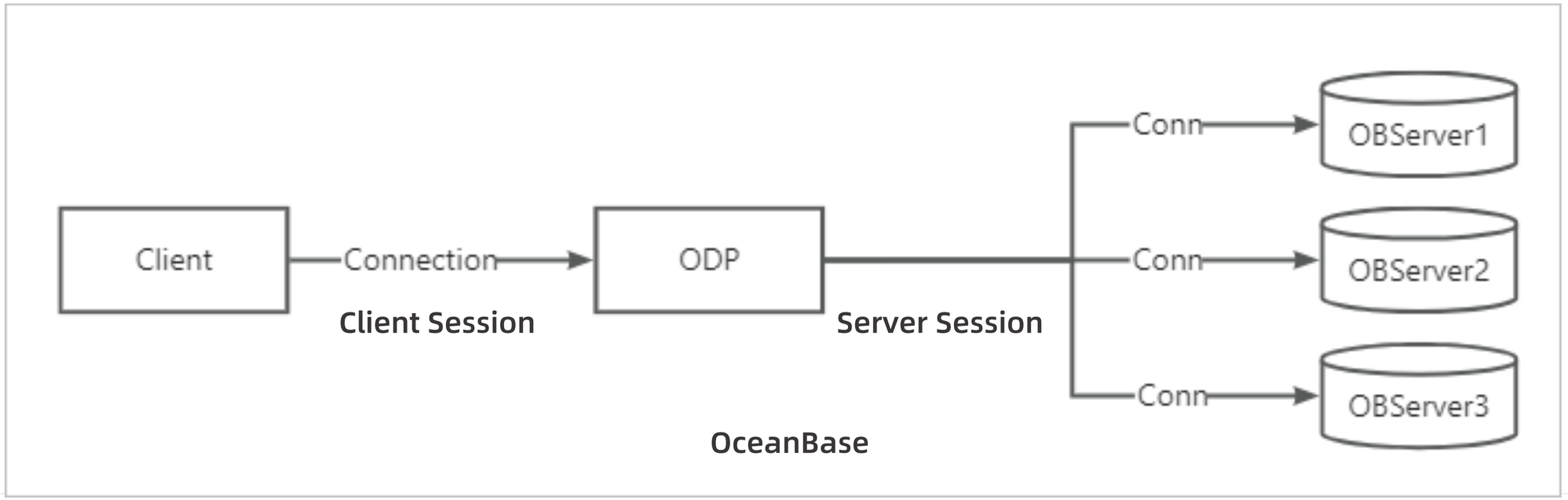
- ODP SQL Routing Principles
- Managing Database Connections
- Analyzing SQL Monitoring Views
- Read and Manage SQL Execution Plans

Lesson 8.2

- Common SQL Tuning Methods
- Troubleshooting Ideas for SQL Performance Issues

Database Connection

Connection Mapping



Database Connection Management

View Client Connections

When connecting through ODP, execute the command under the sys tenant

```
obclient> show proxysession;
+-------------------+---------+-----------------------------+---------+------+---------------------+-----------+-------------+-----------------+-------------------+-------+-------+-----------+
| proxy_sessid      | Id      | Cluster                     | Tenant  | User | Host                | db        | trans_count | svr_session_count | state             | tid   | pid   | using_ssl |
+-------------------+---------+-----------------------------+---------+------+---------------------+-----------+-------------+-----------------+-------------------+-------+-------+-----------+
| 838175694068187151 | 1048577 | obn.xiaofeng.lby.123.456.78.9 | sys     | root | 123.456.78.9:39012 | oceanbase |           0 |                 1 | MCS_ACTIVE_READER | 73180 | 73104 |         0 |
| 838175694068187149 |       5 | obn.xiaofeng.lby.123.456.78.9 | mysql   | root | 123.456.78.9:38027 | test      |           0 |                 1 | MCS_ACTIVE_READER | 73104 | 73104 |         0 |
| 838175694068187150 |  524297 | obn.xiaofeng.lby.123.456.78.9 | mysql   | root | 123.456.78.9:38270 | oceanbase |           0 |                 1 | MCS_ACTIVE_READER | 73179 | 73104 |         0 |
+-------------------+---------+-----------------------------+---------+------+---------------------+-----------+-------------+-----------------+-------------------+-------+-------+-----------+
3 rows in set (0.04 sec)
```

Field	Description
proxy_sessid	The ID number that marks each session with ODP in the OceanBase database
Id	The ID number of each Client Session in ODP, which is cs_id below
trans_count	The number of transactions completed by the client session
svr_session_count	The total number of sessions maintained between ODP and the OceanBase database
state	Client session status, there are several states: • MCS_INIT • MCS_ACTIVE_READER • MCS_KEEP_ALIVE • MCS_HALF_CLOSE • MCS_CLOSED
tid	Thread ID
pid	Process ID
using_ssl	Whether the client session uses the SSL protocol for transmission

Database Connection Management

View Client Connection Details

When connecting through ODP, you can use the "SHOW PROXYSESSION ATTRIBUTE [id [like 'xxx']]" statement to view the detailed internal status of the specified "Client Session", including the related Server Sessions designed on the Client Session.

```
obclient> SHOW PROXYSESSION;
+-------------------+----+---------+--------+------+-----------------+------+-------------+-----------------+----------------+---------+---------+
| proxy_sessid      | Id | Cluster | Tenant | User | Host            | db   | trans_count | svr_session_count | state          | tid     | pid     |
+-------------------+----+---------+--------+------+-----------------+------+-------------+-----------------+----------------+---------+---------+
| 756006681247547396 |  2 | ob1.cc  | sys    | root | 127.0.0.1:22540 | NULL |           0 |               1 | MCS_ACTIVE_READER | 2230520 | 2230520 |
+-------------------+----+---------+--------+------+-----------------+------+-------------+-----------------+----------------+---------+---------+
1 row in set

obclient> SHOW PROXYSESSION ATTRIBUTE;
+-----------------------------+---------------------+-----------------+
| attribute_name              | value               | info            |
+-----------------------------+---------------------+-----------------+
| proxy_sessid                | 756006681247547396  | cs common       |
| cs_id                       | 2                   | cs common       |
| cluster                     | ob1.cc              | cs common       |
| tenant                      | sys                 | cs common       |
| user                        | root                | cs common       |
| host_ip                     | 127.0.0.1           | cs common       |
| host_port                   | 22540               | cs common       |
| db                          | NULL                | cs common       |
| total_trans_cnt             | 0                   | cs common       |
| svr_session_cnt             | 1                   | cs common       |
| active                      | true                | cs common       |
| read_state                  | MCS_ACTIVE_READER   | cs common       |
......

39 rows in set

obclient> SHOW PROXYSESSION ATTRIBUTE 2 like '%id%';
+-----------------------+---------------------+-----------------+
| attribute_name        | value               | info            |
+-----------------------+---------------------+-----------------+
| proxy_sessid          | 756006681247547396  | cs common       |
| cs_id                 | 2                   | cs common       |
| tid                   | 2230520             | cs common       |
| pid                   | 2230520             | cs common       |
| last_insert_id_version | 0                  | cs var version  |
| server_sessid         | 2147549201          | last used ss    |
| ss_id                 | 4                   | last used ss    |
| last_insert_id_version | 0                  | last used ss    |
+-----------------------+---------------------+-----------------+
8 rows in set
```

Database Connection Management

Use the root user to log in to the sys tenant of the OceanBase database to view all servers in the cluster

```
obclient> select * from oceanbase.__all_server;
+----------------------------+----------------------------+-----------+----------+----+------+------------+------------------+--------+
| gmt_create                 | gmt_modified               | svr_ip    | svr_port | id | zone | inner_port | with_rootserver | status |
+----------------------------+----------------------------+-----------+----------+----+------+------------+------------------+--------+
| 2023-02-28 15:45:53.230044 | 2023-02-28 15:46:25.577180 | 10.10.10.1 |     2882 |  3 | z3   |       2881 |                1 | ACTIVE |
| 2023-02-28 15:45:53.197477 | 2023-02-28 15:46:25.534448 | 10.10.10.2 |     2882 |  2 | z2   |       2881 |                0 | ACTIVE |
| 2023-02-28 15:45:53.113870 | 2023-02-28 15:46:25.098607 | 10.10.10.3 |     2882 |  1 | z1   |       2881 |                0 | ACTIVE |
+----------------------------+----------------------------+-----------+----------+----+------+------------+------------------+--------+
3 rows in set
```

Execute the command "show processlist" to view all connections of the current OBServer node

```
[admin@test001 ~]$ obclient -h 10.10.10.1 -P2881 -uroot@sys -p -Doceanbase -A

obclient> show processlist;
+------------+---------+-------------------+-----------+---------+------+--------+--------------------+
| Id         | User    | Host              | db        | Command | Time | State  | Info               |
+------------+---------+-------------------+-----------+---------+------+--------+--------------------+
| 3221812197 | root    | 10.10.10.1:48563 | oceanbase | Query   |    0 | ACTIVE | show processlist |
| 3222117829 | proxyro | 10.10.10.1:37876 | oceanbase | Sleep   |    6 | SLEEP  | NULL               |
| 3221709618 | root    | 10.10.10.1:51390 | oceanbase | Sleep   |  831 | SLEEP  | NULL               |
+------------+---------+-------------------+-----------+---------+------+--------+--------------------+
3 rows in set
```

Agenda

Lesson 8.1

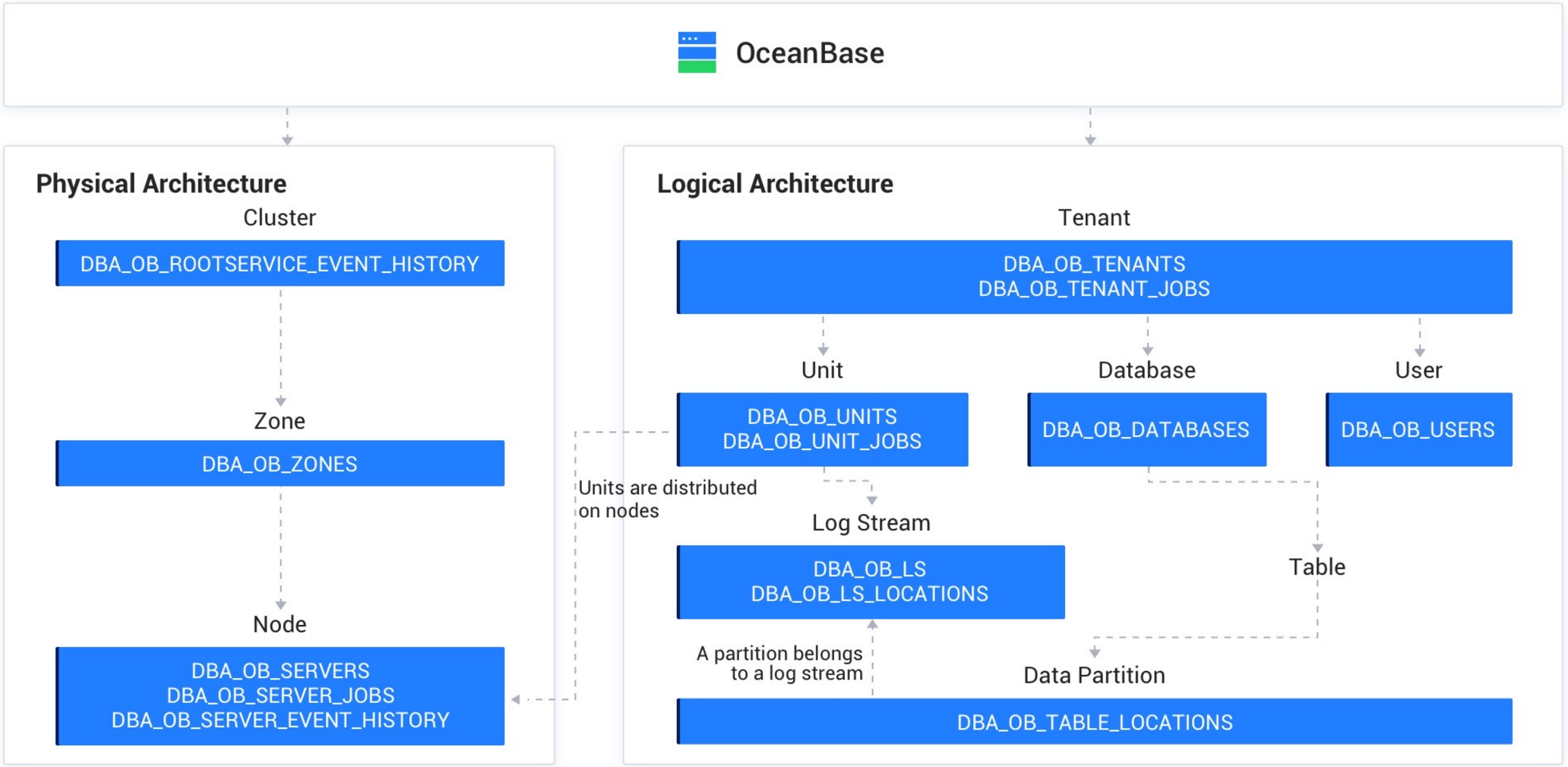
- ODP SQL Routing Principles
- Managing Database Connections
- Analyzing SQL Monitoring Views
- Read and Manage SQL Execution Plans

Lesson 8.2

- Common SQL Tuning Methods
- Troubleshooting Ideas for SQL Performance Issues

SQL Monitoring Views

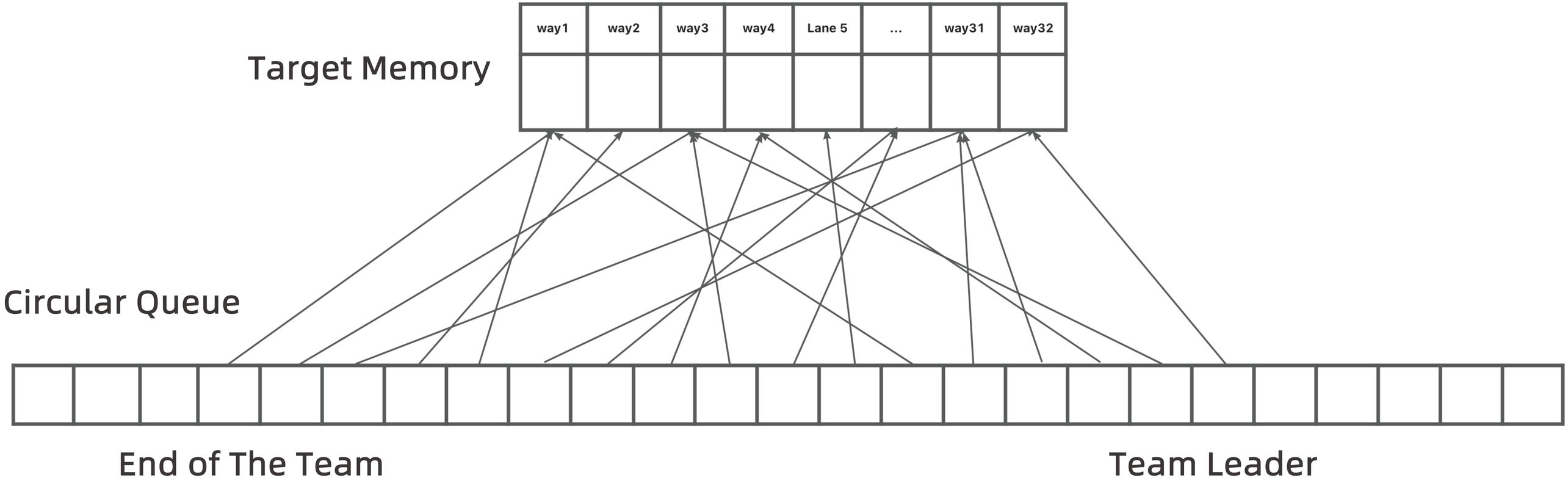
System View Overview



SQL Monitoring Views

Introduction to SQL Audit Monitoring Views

GV$OB_SQL_AUDIT is the most commonly used SQL monitoring view, which can record the source, execution status, resource consumption, and waiting events of each SQL request, regardless of whether the SQL execution is successful or failed. In addition, it also records key information such as SQL text and execution plan. This view is a powerful tool for diagnosing SQL problems.



SQL Monitoring Views

SQL Audit Monitoring View-related Settings

enable_sql_audit: Cluster configuration item, the default value is True, controls whether the SQL Audit function of all tenants is enabled, and takes effect dynamically.

-- View the value of enable_sql_audit

```
show parameters like 'enable_sql_audit'\G
*************************** 1. row ***************************
         zone: zone1
     svr_type: observer
       svr_ip: 1.2.3.4
     svr_port: 12345
         name: enable_sql_audit
    data_type: NULL
        value: True
         info: specifies whether SQL audit is turned on. The default value is TRUE. Value: TRUE: turned on FALSE: turned off
      section: OBSERVER
        scope: CLUSTER
       source: DEFAULT
   edit_level: DYNAMIC_EFFECTIVE
default_value: true
    isdefault: 1
1 row in set (0.09 sec)
```

-- The following two commands need to be logged in and executed using the sys tenant to take effect on the entire cluster

-- Enable SQL Audit for the entire cluster

```
ALTER SYSTEM SET enable_sql_audit = true;
```

-- Disable SQL Audit for the entire cluster

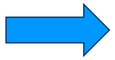
```
ALTER SYSTEM SET enable_sql_audit = false;
```

SQL Monitoring Views

SQL Audit Monitoring View-related Settings

ob_enable_sql_audit:

- Tenant-level system variable
- The default value is ON
- Controls whether the SQL Audit function is enabled for the current tenant
- Dynamically takes effect



```
-- View the value of ob_enable_sql_audit
show variables like 'ob_enable_sql_audit';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| ob_enable_sql_audit | ON    |
+----------------------+-------+
1 row in set (0.09 sec)
```

```
-- Disable the SQL Audit function for the current tenant, which will take effect on newly created connections after the command is successfully executed.
SET global ob_enable_sql_audit = OFF;
```

```
-- Enable the SQL Audit function for the current tenant, which will take effect on newly created connections after the command is successfully executed.
SET global ob_enable_sql_audit = ON;
```

```
-- View the value of ob_sql_audit_percentage
show variables like 'ob_sql_audit_percentage';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| ob_sql_audit_percentage | 3     |
+-------------------------+-------+
1 row in set (0.09 sec)
```

```
-- Adjust the ob_sql_audit_percentage of the current tenant, which will take effect on newly created connections after the command is successfully executed.
SET global ob_sql_audit_percentage = 5;
```

ob_sql_audit_percentage:

- Tenant-level system variable,
- Controls the percentage of tenant memory occupied by the SQL Audit function of the current tenant. The default value is 3 (that is, three percent).
- It takes effect dynamically.

SQL Monitoring Views

SQL Audit Elimination Mechanism

The tenant's background task will decide whether to trigger SQL elimination every 1 second based on the memory usage of the OBServer node and SQL Audit.

Trigger Mechanism	SQL Audit Memory Limit	Conditions for Trigger Elimination	Conditions for Stopping Elimination
Memory Usage	[0, 64MB]	Memory limit * 50%	0 MB
Memory Usage	[64MB, 100MB]	Memory limit - 20 MB	Memory limit - 40 MB
Memory Usage	[100MB, 5GB]	Memory limit * 80%	Memory limit * 60%
Memory Usage	[5GB, +∞)	Memory limit - 1 GB	Memory limit - 2 GB
Number of Records	/	9 million	8 million

In addition, SQL Audit supports manual elimination at the tenant level. The elimination method is to execute the following command under the sys tenant:

```
alter system flush sql audit tenant = tenant_name;
```

SQL Monitoring Views

SQL Audit View Field Description

Only a few fields are selected for introduction here. For a complete introduction to the fields in the GV$OB_SQL_AUDIT view, see: Official Website

TENANT_ID, SVR_ID, SVR_PORT
- Maintain a set of circular arrays to fill and eliminate data

QUERY_SQL, SQL_ID
- Records the execution of SQL and generates 32-bit code based on the SQL

IS_HIT_PLAN, PLAN_ID
- Whether the plan cache is hit, and the corresponding execution plan ID

IS_INNER_SQL, PARAMS_VALUE
- In the PL scenario, SQL is marked as internal SQL
- In the PS scenario, the variable value carried by SQL

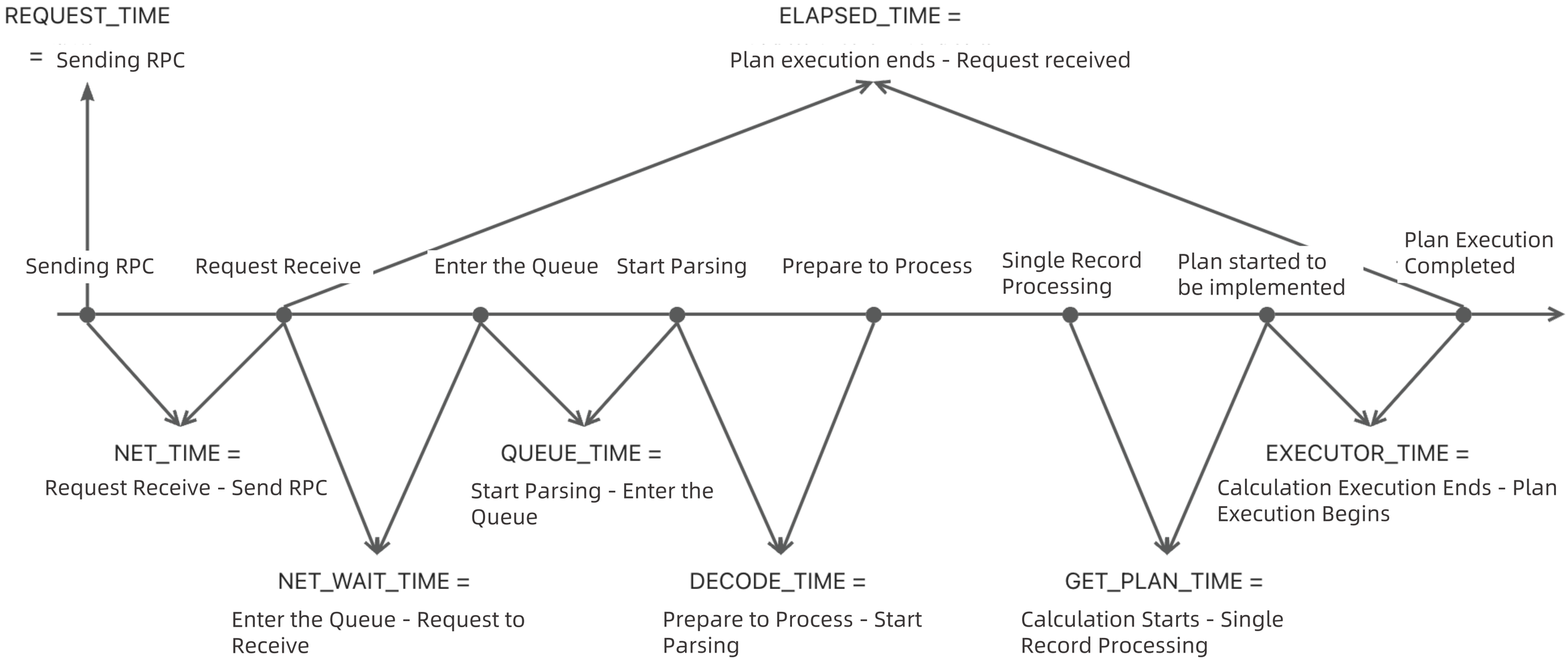
PLAN_TYPE, REQUEST_TYPE
- Schedule type: local, remote, distributed
- Request type: internal, local, remote, distributed, PS, PL

Some important event intervals.

SQL Monitoring Views

SQL Audit View Field Description

Only a few fields are selected for introduction here. For a complete introduction to the fields in the GV$OB_SQL_AUDIT view, see: Official Website



SQL Monitoring Views

GV$OB_SQL_AUDIT View Usage Example

Slow SQL statistics: query the SQL that takes more than a certain threshold within a period of time. You can then tune the specific SQL based on the query results.

- The example queries the tenant with tenant ID 1002 and the top 5 SQL statements with query execution time exceeding 100ms after 2024-02-20 12:00:00.

```
select
  tenant_id,
  request_id,
  usec_to_time(request_time),
  elapsed_time,
  queue_time,
  execute_time,
  query_sql
from
  oceanbase.GV$OB_SQL_AUDIT
where
  tenant_id = 1002
  and elapsed_time > 100000
  and request_time > time_to_usec('2024-02-20 12:00:00')
order by
  elapsed_time desc
limit
  5;
```

```
+-----------+------------+-------------------------------+--------------+------------+--------------+-----------------------------+
| tenant_id | request_id | usec_to_time(request_time)    | elapsed_time | queue_time | execute_time | query_sql                   |
+-----------+------------+-------------------------------+--------------+------------+--------------+-----------------------------+
|      1002 |   21371247 | 2024-02-20 16:14:10.008111 |       153118 |         34 |       139873 | select * from xxxx where xxxx |
+-----------+------------+-------------------------------+--------------+------------+--------------+-----------------------------+
1 row in set (0.11 sec)
```

SQL Monitoring Views

GV$OB_SQL_AUDIT View Usage Example

SQL statistics involved in slow transactions: Each SQL in sql_audit is expected to record the transaction_hash, the unique identifier of the transaction involved in the current SQL.
This field can be used to find all SQL information in the current transaction, and then determine whether the transaction model of the business stress test meets expectations based on this information.

- In this example, the query is for the sys tenant with tenant id 1. Which queries are executed in the transaction tx_id='2759485'

```
select
  tenant_id,
  tx_id,
  query_sql
from
  oceanbase.GV$OB_SQL_AUDIT
where
  tenant_id = 1
  and tx_id = '27592485'
order by
  request_time asc;
```

```
+-----------+----------+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
| tenant_id | tx_id    | query_sql                                                                                                                                                     |
+-----------+----------+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
|         1 | 27592485 | START TRANSACTION                                                                                                                                             |
|         1 | 27592485 | SELECT column_value FROM __all_core_table WHERE TABLE_NAME = '__all_global_stat' AND COLUMN_NAME = 'snapshot_gc_scn' FOR UPDATE                             |
|         1 | 27592485 | UPDATE __all_core_table SET column_value = 1708416843896658521 WHERE table_name = '__all_global_stat' AND column_name = 'snapshot_gc_scn' AND column_value < 1708416843896658521 |
|         1 | 27592485 | COMMIT                                                                                                                                                        |
+-----------+----------+---------------------------------------------------------------------------------------------------------------------------------------------------------------+
4 rows in set (0.11 sec)
```

SQL Monitoring Views

GV$OB_SQL_AUDIT View Usage Example

```
select
  SQL_ID,
  avg(ELAPSED_TIME),
  avg(QUEUE_TIME),
  avg(ROW_CACHE_HIT + BLOOM_FILTER_CACHE_HIT + BLOCK_CACHE_HIT + DISK_READS) avg_logical_read,
  avg(execute_time) avg_exec_time,
  count(*) cnt,
  avg(execute_time - TOTAL_WAIT_TIME_MICRO) avg_cpu_time,
  avg(TOTAL_WAIT_TIME_MICRO) avg_wait_time,
  WAIT_CLASS,
  avg(retry_cnt),
  QUERY_SQL
from
  oceanbase.GV$OB_SQL_AUDIT
group by
  SQL_ID
order by
  avg_exec_time * cnt desc
limit
  10;
```

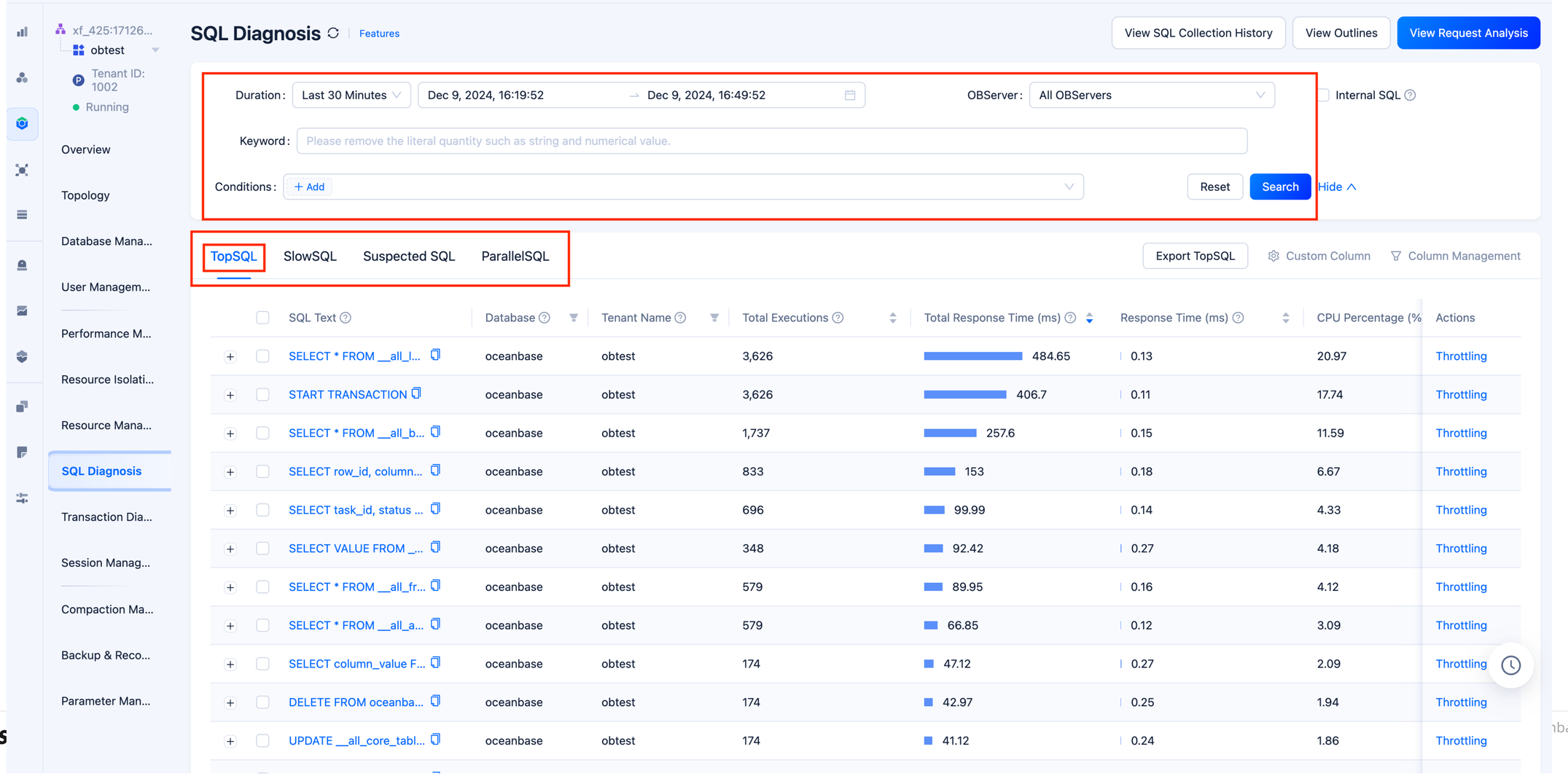
SQL_ID	avg(ELAPSED_TIME)	avg(QUEUE_TIME)	avg_logical_read	avg_exec_time	cnt	avg_cpu_time	avg_wait_time	WAIT_CLASS	avg(retry_cnt)	QUERY_SQL
1532BA78C664771E7113567D8E951B51	4330.2720	0.0000	0.0000	4138.0498	261	4138.0498	0.0000	OTHER	0.0000	SELECT FIELD f
1D0BA376E273B9D622641124D8C59264	323.2321	0.0000	0.0000	224.9437	711	224.9437	0.0000	OTHER	0.0000	COMMIT
9050622DE000F09344C9A882DAF34A75	472.4209	0.0000	0.0000	265.2209	575	265.2209	0.0000	OTHER	0.0000	select * from
19A6B821DB3DE80D69EE8880D3A45C5F	160857.0000	38.0000	0.0000	137214.0000	1	137214.0000	0.0000	OTHER	0.0000	select * from
37B3D4D55DC217FA1BCA9031DC3AF1DC	34309.2500	37.2500	0.0000	28792.0000	4	28792.0000	0.0000	OTHER	0.0000	select SQL_
735537F7B5DB7C4E0E946C9B26108560	615.7653	0.0000	0.0000	395.2419	277	395.2419	0.0000	OTHER	0.0000	SELECT * FROM
2EAAA3C495632AB97F13659E429FC9CD	464.6715	0.0000	0.0000	240.4224	277	240.4224	0.0000	OTHER	0.0000	select * from
B7A6FA97FEC98C06F9586D23935AC4C6	276.7934	0.0000	0.0000	104.0799	576	104.0799	0.0000	OTHER	0.0000	START TRANSACT
556CEFD22F28DDDCB6E283DF89BD9348	2263.9643	0.0000	0.0000	2103.5714	28	2103.5714	0.0000	OTHER	0.0000	UPDATE __all_
17605A1DA6B6A2150E9FBCA5D4C7653A	797.5532	0.0000	0.0000	594.5532	94	594.5532	0.0000	OTHER	0.0000	SELECT row_id,

10 rows in set (0.15 sec)

SQL Monitoring Views

GV$OB_SQL_AUDIT View Usage Example

Slow SQL statistics: query the SQL that takes more than a certain threshold within a period. You can then tune the specific SQL based on the query results.

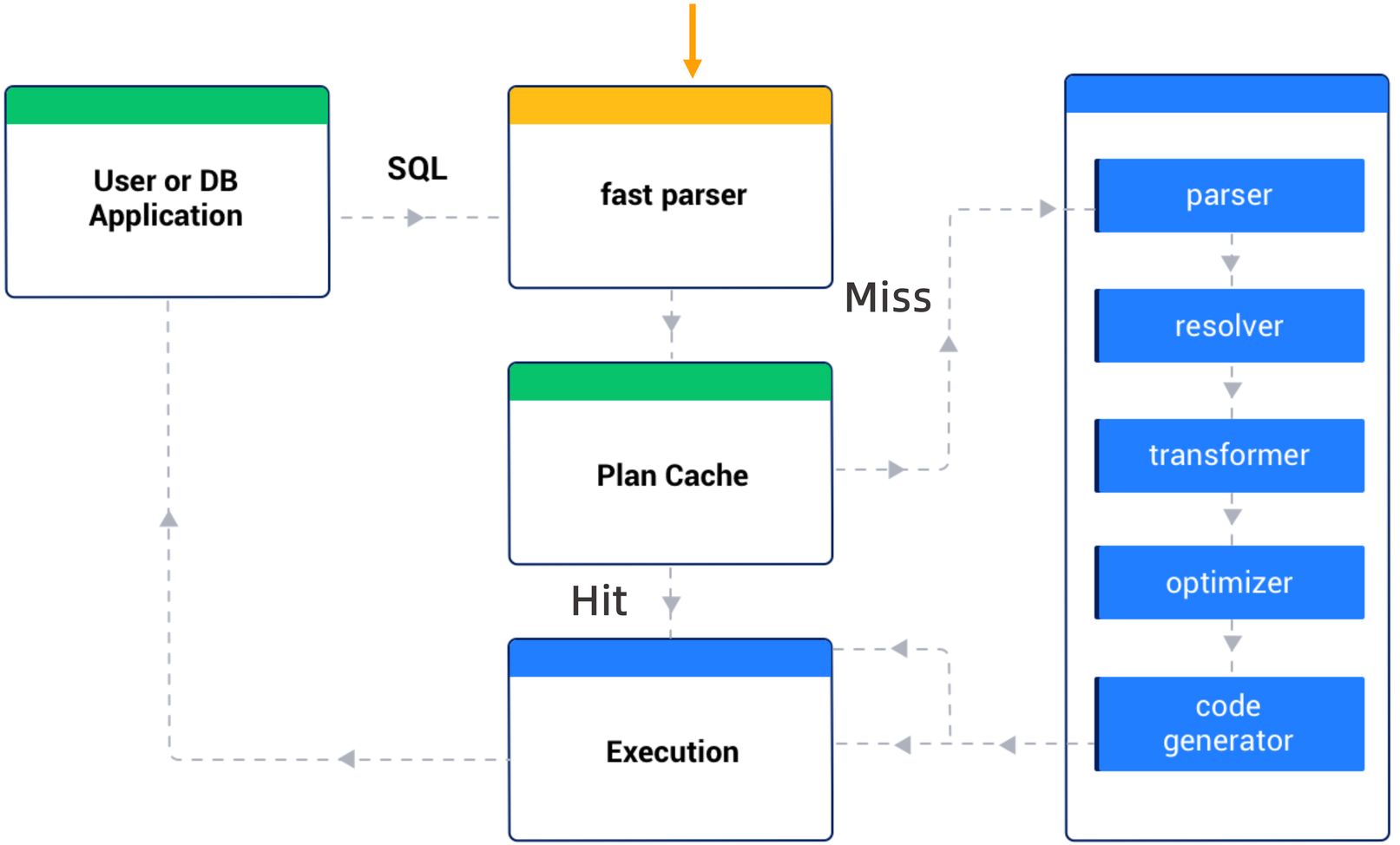


SQL Text	Database	Tenant Name	Total Executions	Total Response Time (ms)	Response Time (ms)	CPU Percentage (%	Actions
SELECT * FROM __all_l...	oceanbase	obtest	3,626	484.65	0.13	20.97	Throttling
START TRANSACTION	oceanbase	obtest	3,626	406.7	0.11	17.74	Throttling
SELECT * FROM __all_b...	oceanbase	obtest	1,737	257.6	0.15	11.59	Throttling
SELECT row_id, column...	oceanbase	obtest	833	153	0.18	6.67	Throttling
SELECT task_id, status ...	oceanbase	obtest	696	99.99	0.14	4.33	Throttling
SELECT VALUE FROM _...	oceanbase	obtest	348	92.42	0.27	4.18	Throttling
SELECT * FROM __all_fr...	oceanbase	obtest	579	89.95	0.16	4.12	Throttling
SELECT * FROM __all_a...	oceanbase	obtest	579	66.85	0.12	3.09	Throttling
SELECT column_value F...	oceanbase	obtest	174	47.12	0.27	2.09	Throttling
DELETE FROM oceanba...	oceanbase	obtest	174	42.97	0.25	1.94	Throttling
UPDATE __all_core_tabl...	oceanbase	obtest	174	41.12	0.24	1.86	Throttling

SQL Monitoring Views

Execution plan monitoring view - plan cache overview

Fast parser performs a fast parameterization on the SQL text. The function of fast parameterization is to replace the constant parameters in the SQL text with wildcards ?
SELECT * FROM t1 WHERE c1 = 1 will be replaced by SELECT * FROM t1 WHERE c1 = ?



SQL Monitoring Views

Execution plan monitoring view GV$OB_PLAN_CACHE_PLAN_EXPLAIN

How can we use these two plan monitoring views to confirm the actual execution plan of a specific SQL statement?

- This example shows how to accurately query select * from oceanbase.DBA_OB_DATABASES; The actual plan of this SQL when it is executed

```
select * from oceanbase.DBA_OB_DATABASES;
+--------------------+---------------+--------------------+-----------+--------------------+
| DATABASE_NAME      | IN_RECYCLEBIN | COLLATION          | READ_ONLY | COMMENT            |
+--------------------+---------------+--------------------+-----------+--------------------+
| obproxy            | NO            | utf8mb4_general_ci | NO        |                    |
| test               | NO            | utf8mb4_general_ci | NO        | test schema        |
| __public           | NO            | utf8mb4_general_ci | YES       | public schema      |
| __recyclebin       | NO            | utf8mb4_general_ci | YES       | recyclebin schema  |
| mysql              | NO            | utf8mb4_general_ci | NO        | MySql schema       |
| information_schema | NO            | utf8mb4_general_ci | NO        | information_schema |
| oceanbase          | NO            | utf8mb4_general_ci | NO        | system database    |
+--------------------+---------------+--------------------+-----------+--------------------+
8 rows in set (0.04 sec)

select last_trace_id();
+-----------------------------------+
| last_trace_id()                   |
+-----------------------------------+
| Y584A0B9E1F14-0006104BFEB3B549-0-0 |
+-----------------------------------+
1 row in set (0.001 sec)
```

SQL Monitoring Views

Execution plan monitoring view GV$OB_PLAN_CACHE_PLAN_EXPLAIN

How can we use these two plan monitoring views to confirm **the actual execution plan** of a specific SQL statement?

- Use oceanbase.gv$ob_sql_audit to query the TENANT_ID, SVR_IP, SVR_PORT, and PLAN_ID fields corresponding to the above SQL
- Because when you query the GV$OB_PLAN_CACHE_PLAN_EXPLAIN view later, you need to specify the equivalent conditions of TENANT_ID, SVR_IP, SVR_PORT, and PLAN_ID, otherwise the query structure will be empty

```
select TENANT_ID,
  SVR_IP,
  SVR_PORT,
  PLAN_ID,
  QUERY_SQL
from oceanbase.gv$ob_sql_audit
where trace_id = 'Y584A0B9E1F14-0006104BFEB3B549-0-0';
+-----------+--------------+----------+---------+----------------------------------------+
| TENANT_ID | SVR_IP       | SVR_PORT | PLAN_ID | QUERY_SQL                              |
+-----------+--------------+----------+---------+----------------------------------------+
|      1002 | 1.2.3.4      |    12345 |   67890 | select * from oceanbase.DBA_OB_DATABASES |
+-----------+--------------+----------+---------+----------------------------------------+
1 row in set (0.06 sec)
```

- Specify the equivalent conditions of TENANT_ID, SVR_IP, SVR_PORT, and PLAN_ID, and query the GV$OB_PLAN_CACHE_PLAN_EXPLAIN view to obtain the actual plan of the corresponding SQL when it is executed.

```
SELECT
  *
FROM
  oceanbase.GV$OB_PLAN_CACHE_PLAN_EXPLAIN
WHERE
  tenant_id = 1002
  AND SVR_IP = '1.2.3.4'
  AND SVR_PORT = 12345
  AND PLAN_ID = 67890;
+-----------+--------------+----------+---------+------------+--------------+----------------+------+------+------+------------------------------------------
| TENANT_ID | SVR_IP       | SVR_PORT | PLAN_ID | PLAN_DEPTH | PLAN_LINE_ID | OPERATOR       | NAME | ROWS | COST | PROPERTY
+-----------+--------------+----------+---------+------------+--------------+----------------+------+------+------+------------------------------------------
|      1002 | 1.2.3.4      |    12345 |   67890 |          0 |            0 | PHY_HASH_JOIN  | NULL |    8 |   56 | NULL
|      1002 | 1.2.3.4      |    12345 |   67890 |          1 |            1 |  PHY_TABLE_SCAN | D    |    8 |    5 | table_rows:8, physical_range_rows:8, logical_range_rows:8, index_back_rows:0, output_rows:8, avaiable_index_
|      1002 | 1.2.3.4      |    12345 |   67890 |          1 |            2 |  PHY_TABLE_SCAN | C    |   18 |   45 | table_rows:18, physical_range_rows:18, logical_range_rows:18, index_back_rows:0, output_rows:18, avaiable_in
+-----------+--------------+----------+---------+------------+--------------+----------------+------+------+------+------------------------------------------
3 rows in set (0.05 sec)
```

SQL Monitoring Views

Execution plan monitoring view GV$OB_PLAN_CACHE_PLAN_EXPLAIN

How can we use these two plan monitoring views to confirm the actual execution plan of a specific SQL statement?

```
show create view oceanbase.DBA_OB_DATABASES;

CREATE VIEW `DBA_OB_DATABASES` AS
SELECT D.DATABASE_NAME AS DATABASE_NAME,
       (CASE D.IN_RECYCLEBIN
            WHEN 0 THEN 'NO'
            ELSE 'YES'
        END) AS IN_RECYCLEBIN,
       C.COLLATION AS COLLATION,
       (CASE D.READ_ONLY
            WHEN 0 THEN 'NO'
            ELSE 'YES'
        END) AS READ_ONLY,
       D.COMMENT AS COMMENT
FROM OCEANBASE.__ALL_DATABASE AS D
LEFT JOIN OCEANBASE.__TENANT_VIRTUAL_COLLATION AS C ON D.COLLATION_TYPE = C.COLLATION_TYPE;
```

Agenda

Lesson 8.1

- ODP SQL Routing Principles
- Managing Database Connections
- Analyzing SQL Monitoring Views
- Read and Manage SQL Execution Plans

Lesson 8.2

- Common SQL Tuning Methods
- Troubleshooting Ideas for SQL Performance Issues

Read Oceanbase SQL Execution Plan

```
create table t1(c1 int, c2 int);

create table t2(c1 int, c2 int);
```

- Insert 10 rows of test data into table t1. The values of column c1 are consecutive integers from 1 to 1000.

```
insert into t1 with recursive cte(n) as (select 1 from dual union all select n + 1 from cte where n < 1000) select n, n from cte;
```

- Insert 10 rows of test data into table t2. The values of column c1 are consecutive integers from 1 to 1000.

```
insert into t2 with recursive cte(n) as (select 1 from dual union all select n + 1 from cte where n < 1000) select n, n from cte;
```

- Collect statistics for the specified table t1

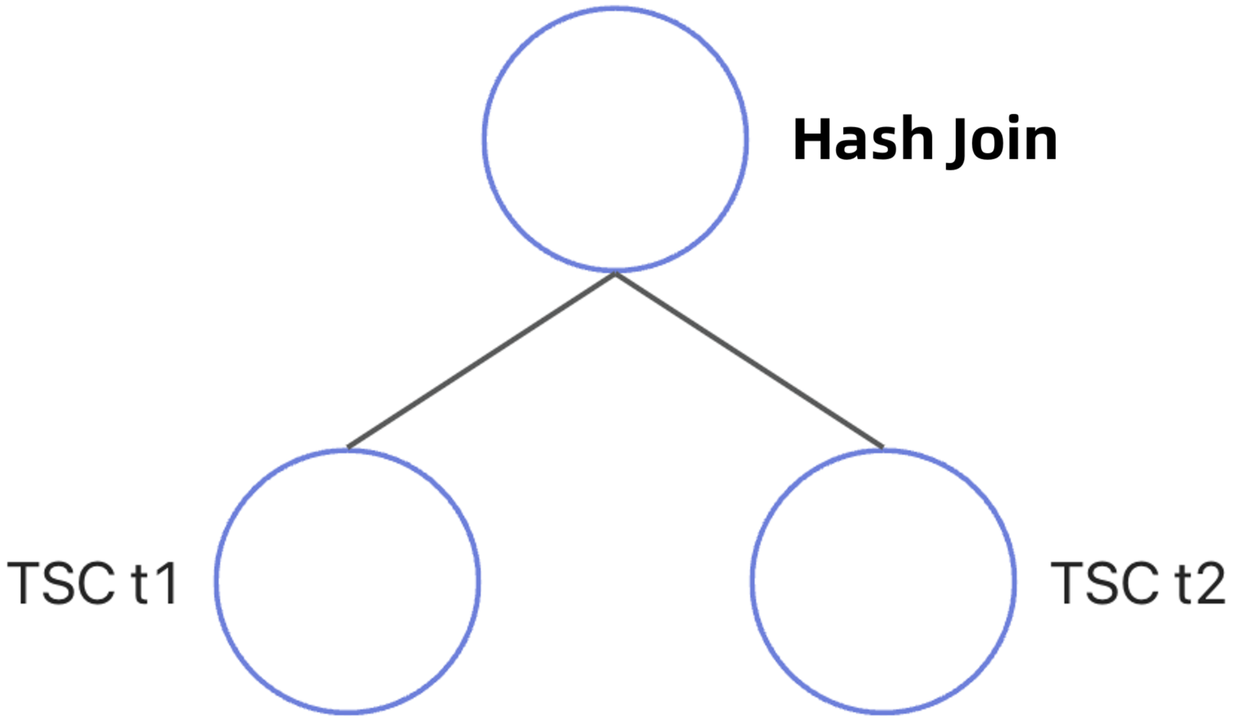
```
analyze table t1 COMPUTE STATISTICS for all columns size 128;
```

- Collect statistics for the specified table t2

```
analyze table t2 COMPUTE STATISTICS for all columns size 128;
```

```
explain select * from t1, t2 where t1.c1 = t2.c1 and t1.c1 < 500;
```

```
+-----------------------------------------------------------------------------------------------+
| Query Plan                                                                                    |
+-----------------------------------------------------------------------------------------------+
| ============================================                                                  |
| |ID|OPERATOR         |NAME|EST.ROWS|EST.TIME(us)|                                            |
| --------------------------------------------                                                  |
| |0 |HASH JOIN        |    |498     |315         |                                            |
| |1 |├─TABLE FULL SCAN|t1  |499     |76          |                                            |
| |2 |└─TABLE FULL SCAN|t2  |499     |76          |                                            |
| ============================================                                                  |
| Outputs & filters:                                                                            |
| -------------------------------------                                                         |
|   0 - output([t1.c1], [t1.c2], [t2.c1], [t2.c2]), filter(nil), rowset=256                     |
|       equal_conds([t1.c1 = t2.c1]), other_conds(nil)                                          |
|   1 - output([t1.c1], [t1.c2]), filter([t1.c1 < 500]), rowset=256                             |
|       access([t1.c1], [t1.c2]), partitions(p0)                                                |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false],             |
|       range_key([t1.__pk_increment]), range(MIN ; MAX)always true                             |
|   2 - output([t2.c1], [t2.c2]), filter([t2.c1 < 500]), rowset=256                             |
|       access([t2.c1], [t2.c2]), partitions(p0)                                                |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false],             |
|       range_key([t2.__pk_increment]), range(MIN ; MAX)always true                             |
+-----------------------------------------------------------------------------------------------+
```



Read OceanBase SQL Execution Plan

```
CREATE TABLE `t1` (
  `c1` int, `c2` int,
  KEY `idx` (`c1`));

insert into t1 values(1, 2);

explain EXTENDED_NOADDR select * from t1 where c1 > 10;
+-----------------------------------------------------------------------------+
| Query Plan                                                                  |
+-----------------------------------------------------------------------------+
| ============================================                                |
| |ID|OPERATOR         |NAME    |EST.ROWS|EST.TIME(us)|                       |
| --------------------------------------------                                |
| |0 |TABLE RANGE SCAN|t1(idx)|1       |7           |                       |
| ============================================                                |
| Outputs & filters:                                                          |
| -------------------------------------                                       |
|   0 - output([t1.c1], [t1.c2]), filter(nil), rowset=16                      |
|       access([t1.__pk_increment], [t1.c1], [t1.c2]), partitions(p0)         |
|       is_index_back=true, is_global_index=false,                            |
|       range_key([t1.c1], [t1.__pk_increment]), range(10,MAX ; MAX,MAX),     |
|       range_cond([t1.c1 > 10])                                              |
| Used Hint:                                                                  |
| -------------------------------------                                       |
|   /*+                                                                       |
|                                                                             |
|   */                                                                        |
| Qb name trace:                                                              |
| -------------------------------------                                       |
|   stmt_id:0, stmt_type:T_EXPLAIN                                            |
|   stmt_id:1, SEL$1                                                          |
| Outline Data:                                                               |
| -------------------------------------                                       |
|   /*+                                                                       |
|       BEGIN_OUTLINE_DATA                                                    |
|       INDEX(@"SEL$1" "test"."t1"@"SEL$1" "idx")                             |
|       OPTIMIZER_FEATURES_ENABLE('4.0.0.0')                                  |
|       END_OUTLINE_DATA                                                      |
|   */                                                                        |
| Optimization Info:                                                          |
| -------------------------------------                                       |
|   t1:                                                                       |
|       table_rows:1                                                          |
|       physical_range_rows:1                                                 |
|       logical_range_rows:1                                                  |
|       index_back_rows:1                                                     |
|       output_rows:1                                                         |
|       table_dop:1                                                           |
|       dop_method:Table DOP                                                  |
|       avaiable_index_name:[idx, t1]                                         |
|       unstable_index_name:[t1]                                              |
|       stats version:0                                                       |
|       dynamic sampling level:0                                              |
|   Plan Type:                                                                |
|       LOCAL                                                                 |
|   Note:                                                                     |
|       Degree of Parallelisim is 1 because of table property                 |
+-----------------------------------------------------------------------------+
```

Read OceanBase SQL Execution Plan

Optimization Info - Part I

Optimization Info	Meaning
table_rows	The number of rows in the last merged version of the SSTable can be simply understood as the number of rows in the t1 table and is only for reference.
physical_range_rows	The number of physical rows that need to be scanned in table t1. If an index is used, it means the number of physical rows that need to be scanned in table t1 on the index.
logical_range_rows	The number of logical rows that need to be scanned in table t1. If an index is used, it means the number of logical rows that need to be scanned in table t1 on the index. In the above plan, because the idx index is scanned, you can see that the scan range is range(10,MA×;MAX,MAX), range_cond[t1.c1>101). If there is no index, the entire table needs to be scanned, and the scan range will become range(MIN;MAX). Note: The two indicators physical_range_rows and logical_range_rows are generally similar, and you can look at either one. Only in some special Buffer table scenarios, physical_range_rows may be much larger than logical_range_rows. Buffer tables represent tables with frequent insertions and deletions. When a large amount of deleted data is accumulated in the incremental data in the LSMtree, there are few actual rows from the perspective of the upper-level application, but a large amount of deleted data may need to be processed during range queries. In this scenario, physical_range_rows may be much larger than logical_range_rows, resulting in less than ideal SQL time consumption. At the same time, the Buffer table scenario is also prone to cause the optimizer to generate a non-optimal execution plan. For a detailed introduction to the Buffer table, detection logic, and avoidance methods, see https://en.oceanbase.com/docs/common-oceanbase-database-10000000001718200
index_back_rows	If the index needs to be returned to the table, this value indicates the number of rows returned to the table by the index. When the full table is scanned or the index is scanned but no return to the table is required, this value is 0. Please refer to the official website for the concept of index return to the table. Simply put, the reason why the index needs to be returned to the main table in this plan is that the index idx only contains the data of column c1. It is necessary to match the data filtered out from the index to the hidden primary key information in the main table and return to the main table to find the data of column c2. If this query is not select * from t1 where c1>10, but select c1 from t1 where c1>10, because the index already contains all the information that needs to be queried, there is no need to return to the table by the index.

Read OceanBase SQL Execution Plan

Optimization Info - Part II

Optimization Info	Meaning
output_rows	Estimated number of output rows. In the above plan, it indicates the number of rows in table t1 after filtering.
table_dop	The degree of parallelism when scanning the t1 table (the number of worker threads used for parallel execution)
dop_method	The reason that determines the parallelism of the table scan. It can be TableDOP (the parallelism specified when the table is defined), AutoDop (the parallelism selected by the optimizer based on the cost, the auto dop function needs to be turned on), or global parallel (the parallelism set by parallel hint or system variables)
avaiable_index_name	List of indexes available for table t1. In addition to the index table, this also includes the main table. If there is no suitable index, the plan will choose to perform a full table scan on the main table.
pruned_index_name	The current query is based on the optimizer's rules and is considered to be a list of indexes that should not be used. For details on the index pruning rules of the OceanBase optimizer, see the official website documentation.
unstable_index_name	If this exists, it must be the main table path that is pruned. It is usually pruned because there are other index paths with smaller range rows.
stats version	The t1 table statistics version number. If the value is 0, it means that the table has not collected statistics. To ensure that the plan is generated correctly, you can automatically or manually collect statistics for the corresponding table.
dynamic sampling level	Dynamic sampling level. If the value is 0, it means that the table does not use dynamic sampling. Dynamic sampling is an optimization tool of the optimizer. For details, see the official website documentation
estimation method	The row count estimation method of the t1 table can be DEFAULT (using the default statistical information, in which case the row count estimation may be inaccurate and requires DBA intervention for optimization), STORAGE (using the storage layer to estimate the number of rows in real time), or STATS (using statistical information to estimate the number of rows)
Plan Type	The current plan type can be LOCAL, REMOTE, or DISTRIBUTED.
Note	Some notes about the plan. For example, in the plan above, "Degree of Parallelism is 1 because of table property" means that the degree of parallelism of the current query is set to 1 because the degree of parallelism of the current table is set to 1.

Read OceanBase SQL Execution Plan

```
explain EXTENDED select * from t1 where c1 > 10;
+------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                           |
+------------------------------------------------------------------------------------------------------+
| ====================================================                                                 |
| |ID|OPERATOR         |NAME    |EST.ROWS|EST.TIME(us)|                                                |
| ----------------------------------------------------                                                 |
| |0 |TABLE RANGE SCAN|t1(idx)|1       |7           |                                                  |
| ====================================================                                                 |
| Outputs & filters:                                                                                   |
| -------------------------------------                                                                |
|   0 - output([t1.c1(0x7fed4fe0df50)], [t1.c2(0x7fed4fe0e4c0)]), filter(nil), rowset=16              |
|       access([t1.__pk_increment(0x7fed4fe0e9d0)], [t1.c1(0x7fed4fe0df50)], [t1.c2(0x7fed4fe0e4c0)]), partitions(p0) |
|       is_index_back=true, is_global_index=false,                                                     |
|       range_key([t1.c1(0x7fed4fe0df50)], [t1.__pk_increment(0x7fed4fe0e9d0)]), range(10,MAX ; MAX,MAX), |
|       range_cond([t1.c1(0x7fed4fe0df50) > 10(0x7fed4fe0d800)])                                       |
|                                                                                                      |
|       ......                                                                                         |
|                                                                                                      |
+------------------------------------------------------------------------------------------------------+
```

Read OceanBase SQL Execution Plan

For common operators, see: Execution Plan Operators in the official website documentation.

It is strongly recommended that users learn about the most common TABLE SCAN operators and JOIN operators, as well as DAS (data access service) execution in TABLE SCAN.

Read OceanBase SQL Execution Plan

```
create table t1(c1 int primary key, c2 int, c3 int);

create index idx on t1(c2);

explain select * from t1 where c1 = 1;
+----------------------------------------------------------------------+
| Query Plan                                                           |
+----------------------------------------------------------------------+
| ========================================                             |
| |ID|OPERATOR |NAME|EST.ROWS|EST.TIME(us)|                           |
| ----------------------------------------                             |
| |0 |TABLE GET|t1  |1       |3           |                           |
| ========================================                             |
| Outputs & filters:                                                   |
| -------------------------------------                                |
|   0 - output([t1.c1], [t1.c2], [t1.c3]), filter(nil), rowset=16      |
|       access([t1.c1], [t1.c2], [t1.c3]), partitions(p0)              |
|       is_index_back=false, is_global_index=false,                    |
|       range_key([t1.c1]), range[1 ; 1],                              |
|       range_cond([t1.c1 = 1])                                        |
+----------------------------------------------------------------------+
12 rows in set (0.039 sec)
```

Read OceanBase SQL Execution Plan

```
explain select c2 from t1 where c2 = 1;
+----------------------------------------------------------------------+
| Query Plan                                                           |
+----------------------------------------------------------------------+
| ============================================                         |
| |ID|OPERATOR         |NAME    |EST.ROWS|EST.TIME(us)|               |
| --------------------------------------------                         |
| |0 |TABLE RANGE SCAN |t1(idx) |1       |4           |               |
| ============================================                         |
| Outputs & filters:                                                   |
| -------------------------------------                                |
|   0 - output([t1.c2]), filter(nil), rowset=16                        |
|       access([t1.c2]), partitions(p0)                                |
|       is_index_back=false, is_global_index=false,                    |
|       range_key([t1.c2], [t1.c1]), range(1,MIN ; 1,MAX),             |
|       range_cond([t1.c2 = 1])                                        |
+----------------------------------------------------------------------+
12 rows in set (0.009 sec)
```

Read OceanBase SQL Execution Plan

```
explain select * from t1 where c2 = 1;
+-----------------------------------------------------------------------+
| Query Plan                                                            |
+-----------------------------------------------------------------------+
| ============================================                          |
| |ID|OPERATOR         |NAME    |EST.ROWS|EST.TIME(us)|                |
| --------------------------------------------                          |
| |0 |TABLE RANGE SCAN |t1(idx) |1       |7           |                |
| ============================================                          |
| Outputs & filters:                                                    |
| -------------------------------------                                 |
|   0 - output([t1.c1], [t1.c2], [t1.c3]), filter(nil), rowset=16       |
|       access([t1.c1], [t1.c2], [t1.c3]), partitions(p0)               |
|       is_index_back=true, is_global_index=false,                      |
|       range_key([t1.c2], [t1.c1]), range(1,MIN ; 1,MAX),              |
|       range_cond([t1.c2 = 1])                                         |
+-----------------------------------------------------------------------+
12 rows in set (0.045 sec)
```

Read OceanBase SQL Execution Plan

```
explain select * from t1 where c3 = 1;
+-----------------------------------------------------------------------------+
| Query Plan                                                                  |
+-----------------------------------------------------------------------------+
| ============================================                                |
| |ID|OPERATOR       |NAME|EST.ROWS|EST.TIME(us)|                            |
| --------------------------------------------                                |
| |0 |TABLE FULL SCAN|t1  |1       |4           |                            |
| ============================================                                |
| Outputs & filters:                                                          |
| -------------------------------------                                       |
|   0 - output([t1.c1], [t1.c2], [t1.c3]), filter([t1.c3 = 1]), rowset=16     |
|       access([t1.c1], [t1.c3], [t1.c2]), partitions(p0)                     |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false], |
|       range_key([t1.c1]), range(MIN ; MAX)always true                       |
+-----------------------------------------------------------------------------+
11 rows in set (0.009 sec)
```

Read OceanBase SQL Execution Plan

```
CREATE TABLE t2(c1 INT PRIMARY KEY, c2 INT, c3 INT) PARTITION BY HASH(c1) PARTITIONS 4;

CREATE INDEX i2 ON t2(c2) GLOBAL;

EXPLAIN SELECT * FROM t2 WHERE c2 = 1;
+----------------------------------------------------------------------+
| Query Plan                                                           |
+----------------------------------------------------------------------+
| ============================================================         |
| |ID|OPERATOR                       |NAME   |EST.ROWS|EST.TIME(us)|    |
| ------------------------------------------------------------         |
| |0 |DISTRIBUTED TABLE RANGE SCAN|t2(i2)|1       |30          |         |
| ============================================================         |
| Outputs & filters:                                                   |
| -------------------------------------                                |
|   0 - output([t2.c1], [t2.c2], [t2.c3]), filter(nil), rowset=16      |
|       access([t2.c1], [t2.c2], [t2.c3]), partitions(p0)              |
|       is_index_back=true, is_global_index=true,                      |
|       range_key([t2.c2], [t2.c1]), range(1,MIN ; 1,MAX),             |
|       range_cond([t2.c2 = 1])                                        |
+----------------------------------------------------------------------+
12 rows in set (0.121 sec)
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan Through Hint - Hint Syntax

Hint Grammar：

{ DELETE | INSERT | SELECT | UPDATE | REPLACE } /*+ [hint_text][,hint_text]... */ ...

```
create t1 (c1 int, c2 int) PARTITION BY HASH(c1) PARTITIONS 5;

explain EXTENDED_NOADDR select /*+ PARALLEL(3) */* from t1 where c1 > 10;
+----------------------------------------------------------------------+
| Query Plan                                                           |
+----------------------------------------------------------------------+
| ============================================================         |
| |ID|OPERATOR                |NAME    |EST.ROWS|EST.TIME(us)|         |
| ------------------------------------------------------------         |
| |0 |PX COORDINATOR          |        |1       |3           |         |
| |1 |└─EXCHANGE OUT DISTR    |:EX10000|1       |3           |         |
| |2 |  └─PX BLOCK ITERATOR   |        |1       |3           |         |
| |3 |    └─TABLE RANGE SCAN  |t1(idx) |1       |3           |         |
| ============================================================         |
| Outputs & filters:                                                   |
| -------------------------------------                                |
|   0 - output([INTERNAL_FUNCTION(t1.c1, t1.c2)]), filter(nil), rowset=16 |
|   1 - output([INTERNAL_FUNCTION(t1.c1, t1.c2)]), filter(nil), rowset=16 |
|       dop=3                                                          |
|   2 - output([t1.c1], [t1.c2]), filter(nil), rowset=16               |
|   3 - output([t1.c1], [t1.c2]), filter(nil), rowset=16               |
|       access([t1.__pk_increment], [t1.c1], [t1.c2]), partitions(p0)  |
|       is_index_back=true, is_global_index=false,                     |
|       range_key([t1.c1], [t1.__pk_increment]), range(10,MAX ; MAX,MAX), |
|       range_cond([t1.c1 > 10])                                       |
| Used Hint:                                                           |
| -------------------------------------                                |
|   /*+                                                                |
|       PARALLEL(3)                                                    |
|   */                                                                 |
|                                                                      |
|   ......                                                             |
|                                                                      |
|   Note:                                                              |
|       Degree of Parallelism is 3 because of hint                     |
+----------------------------------------------------------------------+
```

Hint Names	Parameter	Description
NO_REWRITE		Indicates not to rewrite the SQL statement.
READ_CONSISTENCY	weak|strong|frozen	• weak: weak consistency read. • strong: strong consistency read. • frozen: reads data from the last major freeze.
INDEX_HINT	[qb_name] table_nameindex_name	Specifies the index used during a query on a table.
QUERY_TIMEOUT	int64	Specifies the statement execution timeout value, in microseconds (μs).
LEADING	[qb_name] table_name [, table_name]	Specifies the order in which multiple tables are joined.
ORDERED		Specifies that the tables are joined in the order displayed in the SQL statement.
FULL	[qb_name] table_name	Specifies a full table scan as the access method (reads the primary key if any).
USE_MERGE	[qb_name] table_name [,table_name]	Specifies to use the MERGE algorithm for a multi-table join.
USE_NL	[qb_name] table_name [, table_name]	Specifies to use the NEST LOOP algorithm for a multi-table join.

Hint Names	Parameter	Description
USE_BNL	[qb_name] table_name [,table_name]	Specifies to use the BLOCK NEST LOOP algorithm for a multi-table join.
USE_HASH_AGGREGATION	[qb_name]	Specifies to use the HASH AGGREGATE method, such as HASH GROUP BY and HASH DISTINCT, as the aggregate method.
NO_USE_HASH_AGGREGATION	[qb_name]	Specifies to use the MERGE GROUP BY and MERGE DISTINCT methods for aggregate, instead of HASH AGGREGATE.
QB_NAME	[qb_name]	Specifies the name of the query block.
PARALLEL	int64	Specifies the degree of parallelism (DOP) of distributed execution.

For more information, see https://en.oceanbase.com/docs/common-oceanbase-database-10000000001784716

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - QB_NAME (Query Block Name) Parameter

```
CREATE TABLE t1(c1 INT, c2 INT, KEY t1_c1(c1));

CREATE TABLE t2(c1 INT, c2 INT, KEY t2_c1(c1));

EXPLAIN EXTENDED_NOADDR SELECT * FROM t1, (SELECT * FROM t2 WHERE c2 = 1 LIMIT 5) WHERE t1.c1 = 1;
```

```
+----------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                               |
+----------------------------------------------------------------------------------------------------------+
| ============================================================================                             |
| |ID|OPERATOR                   |NAME           |EST.ROWS|EST.TIME(us)|                                 |
| ----------------------------------------------------------------------------                             |
| |0 |NESTED-LOOP JOIN CARTESIAN |               |1       |11          |                                 |
| |1 |├─TABLE RANGE SCAN         |t1(t1_c1)      |1       |7           |                                 |
| |2 |└─MATERIAL                 |               |1       |4           |                                 |
| |3 |  └─SUBPLAN SCAN           |ANONYMOUS_VIEW1|1       |4           |                                 |
| |4 |    └─TABLE FULL SCAN      |t2             |1       |4           |                                 |
| ============================================================================                             |
| Outputs & filters:                                                                                       |
| -------------------------------------                                                                    |
|   0 - output([t1.c1], [t1.c2], [.c1], [.c2]), filter(nil), rowset=16                                     |
|       conds(nil), nl_params_(nil), use_batch=false                                                       |
|   1 - output([t1.c1], [t1.c2]), filter(nil), rowset=16                                                   |
|       access([t1.__pk_increment], [t1.c1], [t1.c2]), partitions(p0)                                      |
|       is_index_back=true, is_global_index=false,                                                         |
|       range_key([t1.c1], [t1.__pk_increment]), range(1,MIN ; 1,MAX),                                     |
|       range_cond([t1.c1 = 1])                                                                            |
|   2 - output([.c1], [.c2]), filter(nil), rowset=16                                                       |
|   3 - output([.c1], [.c2]), filter(nil), rowset=16                                                       |
|       access([.c1], [.c2])                                                                               |
|   4 - output([t2.c1], [t2.c2]), filter([t2.c2 = 1]), rowset=16                                           |
|       access([t2.c2], [t2.c1]), partitions(p0)                                                           |
|       limit(5), offset(nil), is_index_back=false, is_global_index=false, filter_before_indexback[false], |
|       range_key([t2.__pk_increment]), range(MIN ; MAX)always true                                        |
```

Query Block SEL$1 is the outermost: SELECT * FROM t1, VIEW1 WHERE t1.c1 = 1。
Query Block SEL$2 is VIEW1 (i.e. ANONYMOUS_VIEW1 in the plan): SELECT * FROM t2 WHERE c2 = 1 LIMIT 5。

```
| Qb name trace:
| ------------------------------------
|   stmt_id:0, stmt_type:T_EXPLAIN
|   stmt_id:1, SEL$1
|   stmt_id:2, SEL$2
| Outline Data:
| ------------------------------------
|   /*+
|       BEGIN_OUTLINE_DATA
|       LEADING(@"SEL$1" ("test"."t1"@"SEL$1" "ANONYMOUS_VIEW1"@"SEL$1"))
|       USE_NL(@"SEL$1" "ANONYMOUS_VIEW1"@"SEL$1")
|       USE_NL_MATERIALIZATION(@"SEL$1" "ANONYMOUS_VIEW1"@"SEL$1")
|       INDEX(@"SEL$1" "test"."t1"@"SEL$1" "t1_c1")
|       FULL(@"SEL$2" "test"."t2"@"SEL$2")
|       OPTIMIZER_FEATURES_ENABLE('4.0.0.0')
|       END_OUTLINE_DATA
|   */
|   ......
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - QB_NAME (Query Block Name) Parameter

```
EXPLAIN EXTENDED_NOADDR
SELECT /*+ INDEX(@SEL$1 t1 PRIMARY) INDEX(@SEL$2 t2 t2_c1) */ *
FROM t1 , (SELECT * FROM t2 WHERE c2 = 1 LIMIT 5) WHERE t1.c1 = 1;
```

Query Block SEL$1 is the outermost: SELECT * FROM t1, VIEW1 WHERE t1.c1 = 1。
Query Block SEL$2 is VIEW1 (i.e. ANONYMOUS_VIEW1 in the plan): SELECT * FROM t2 WHERE c2 = 1 LIMIT 5。

```
Query Plan
==========================================================================
|ID|OPERATOR                   |NAME           |EST.ROWS|EST.TIME(us)|
--------------------------------------------------------------------------
|0 |NESTED-LOOP JOIN CARTESIAN |               |1       |11          |
|1 |├─TABLE FULL SCAN          |t1             |1       |4           |
|2 |└─MATERIAL                 |               |1       |7           |
|3 |  └─SUBPLAN SCAN           |ANONYMOUS_VIEW1|1       |7           |
|4 |    └─TABLE FULL SCAN      |t2(t2_c1)      |1       |7           |
==========================================================================
Outputs & filters:
-------------------------------------
  0 - output([t1.c1], [t1.c2], [.c1], [.c2]), filter(nil), rowset=16
      conds(nil), nl_params_(nil), use_batch=false
  1 - output([t1.c1], [t1.c2]), filter([t1.c1 = 1]), rowset=16
      access([t1.c1], [t1.c2]), partitions(p0)
      is_index_back=false, is_global_index=false, filter_before_indexback[false],
      range_key([t1.__pk_increment]), range(MIN ; MAX)always true
  2 - output([.c1], [.c2]), filter(nil), rowset=16
  3 - output([.c1], [.c2]), filter(nil), rowset=16
      access([.c1], [.c2])
  4 - output([t2.c1], [t2.c2]), filter([t2.c2 = 1]), rowset=16
      access([t2.__pk_increment], [t2.c2], [t2.c1]), partitions(p0)
      limit(5), offset(nil), is_index_back=true, is_global_index=false, filter_before_indexback[false],
      range_key([t2.c1], [t2.__pk_increment]), range(MIN,MIN ; MAX,MAX)always true
```

```
Used Hint:
------------------------------------
  /*+

      INDEX("t1" "primary")
      INDEX(@"SEL$2" "t2" "t2_c1")
  */
Qb name trace:
------------------------------------
  stmt_id:0, stmt_type:T_EXPLAIN
  stmt_id:1, SEL$1
  stmt_id:2, SEL$2
Outline Data:
------------------------------------
  /*+
      BEGIN_OUTLINE_DATA
      LEADING(@"SEL$1" ("test"."t1"@"SEL$1" "ANONYMOUS_VIEW1"@"SEL$1"))
      USE_NL(@"SEL$1" "ANONYMOUS_VIEW1"@"SEL$1")
      USE_NL_MATERIALIZATION(@"SEL$1" "ANONYMOUS_VIEW1"@"SEL$1")
      FULL(@"SEL$1" "test"."t1"@"SEL$1")
      INDEX(@"SEL$2" "test"."t2"@"SEL$2" "t2_c1")
      OPTIMIZER_FEATURES_ENABLE('4.0.0.0')
      END_OUTLINE_DATA
  */
  ......
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - QB_NAME (Query Block Name) Parameter

Hint in the above example can also be written in the following three ways, which are equivalent:

```
SELECT /*+INDEX(@SEL$1 t1 PRIMARY) INDEX(@SEL$2 t2 t2_c1)*/ * FROM t1 , (SELECT * FROM t2 WHERE c2 = 1 LIMIT 5) WHERE t1.c1 = 1;

<==>

SELECT /*+INDEX(t1 PRIMARY) INDEX(@SEL$2 t2@SEL$2 t2_c1)*/ * FROM t1 , (SELECT * FROM t2 WHERE c2 = 1 LIMIT 5) WHERE t1.c1 = 1;

<==>

SELECT /*+INDEX(t1 PRIMARY)*/ * FROM t1 , (SELECT /*+INDEX(t2 t2_c1)*/ * FROM t2 WHERE c2 = 1 LIMIT 5) WHERE t1.c1 = 1;
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint —— Hint Usage Rules

For a Hint without a specified Query Block, it indicates that it applies to this Query Block:

```
EXPLAIN SELECT /*+INDEX(t2 t2_c1)*/ *
        FROM t1 , (SELECT * FROM t2 WHERE c2 = 1 LIMIT 5)
        WHERE t1.c1 = 1;
```

```
+-----------------------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                                            |
+-----------------------------------------------------------------------------------------------------------------------+
| ============================================================================                                          |
| |ID|OPERATOR                      |NAME            |EST.ROWS|EST.TIME(us)|                                          |
| ----------------------------------------------------------------------------                                          |
| |0 |NESTED-LOOP JOIN CARTESIAN |                |1       |11          |                                          |
| |1 |├─TABLE RANGE SCAN          |t1(t1_c1)       |1       |7           |                                          |
| |2 |└─MATERIAL                  |                |1       |4           |                                          |
| |3 |  └─SUBPLAN SCAN            |ANONYMOUS_VIEW1 |1       |4           |                                          |
| |4 |    └─TABLE FULL SCAN       |t2              |1       |4           |                                          |
| ============================================================================                                          |
| ......                                                                                                                |
+-----------------------------------------------------------------------------------------------------------------------+
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hint INDEX Hint

SELECT/*+ INDEX(table_name index_name) */ * FROM table_name;

```
create table t1(c1 int, c2 int, c3 int);

create index idx1 on t1(c1);

create index idx2 on t1(c2);
```

- Insert 1000 rows of test data

```
insert into t1 with recursive cte(n) as (select 1 from dual union all select n + 1 from cte where n < 1000) select n, mod(n, 3), n from cte;
```

- Collect statistics for the specified table t1

```
analyze table t1 COMPUTE STATISTICS for all columns size 128;
```

- For the data features of these 1000 test data rows, c1=1 has better filtering performance than c2=1
- Therefore, when there is no index hint or the hint is not effective, the optimizer will give priority to using the idx1 index by default when generating a plan.

```
explain select * from t1 where c1 = 1 and c2 = 1;
```

```
+-----------------------------------------------------------------------------------+
| Query Plan                                                                        |
+-----------------------------------------------------------------------------------+
| ==================================================                                |
| |ID|OPERATOR        |NAME    |EST.ROWS|EST.TIME(us)|                              |
| --------------------------------------------------                                |
| |0 |TABLE RANGE SCAN|t1(idx1)|1       |7           |                              |
| ==================================================                                |
| Outputs & filters:                                                                |
| -------------------------------------                                             |
|   0 - output([t1.c1], [t1.c2], [t1.c3]), filter([t1.c2 = 1]), rowset=16           |
|       access([t1.__pk_increment], [t1.c1], [t1.c2], [t1.c3]), partitions(p0)      |
|       is_index_back=true, is_global_index=false, filter_before_indexback[false],  |
|       range_key([t1.c1], [t1.__pk_increment]), range(1,MIN ; 1,MAX),              |
|       range_cond([t1.c1 = 1])                                                     |
+-----------------------------------------------------------------------------------+
```

- Effective index hint

```
explain select /*+index(t idx2)*/ * from t1 as t where c1 = 1 and c2 = 1;
+----------------------------------------------------------------------------------------+
| Query Plan                                                                             |
+----------------------------------------------------------------------------------------+
| ==================================================                                     |
| |ID|OPERATOR        |NAME    |EST.ROWS|EST.TIME(us)|                                     |
| --------------------------------------------------                                     |
| |0 |TABLE RANGE SCAN|t(idx2)|1       |871         |                                     |
| ==================================================                                     |
| Outputs & filters:                                                                     |
| -------------------------------------                                                  |
|   0 - output([t.c1], [t.c2], [t.c3]), filter([t.c1 = 1]), rowset=16                    |
|       access([t.__pk_increment], [t.c1], [t.c2], [t.c3]), partitions(p0)               |
|       is_index_back=true, is_global_index=false, filter_before_indexback[false],       |
|       range_key([t.c2], [t.__pk_increment]), range(1,MIN ; 1,MAX),                     |
|       range_cond([t.c2 = 1])                                                           |
+----------------------------------------------------------------------------------------+
```

- Invalid index because t1 has been assigned an alias t

```
explain select /*+index(t1 idx2)*/ * from t1 t where c1 = 1 and c2 = 1;
+----------------------------------------------------------------------------------------+
| Query Plan                                                                             |
+----------------------------------------------------------------------------------------+
| ==================================================                                     |
| |ID|OPERATOR        |NAME    |EST.ROWS|EST.TIME(us)|                                     |
| --------------------------------------------------                                     |
| |0 |TABLE RANGE SCAN|t(idx1)|1       |7           |                                     |
| ==================================================                                     |
| Outputs & filters:                                                                     |
| -------------------------------------                                                  |
|   0 - output([t.c1], [t.c2], [t.c3]), filter([t.c2 = 1]), rowset=16                    |
|       access([t.__pk_increment], [t.c1], [t.c2], [t.c3]), partitions(p0)               |
|       is_index_back=true, is_global_index=false, filter_before_indexback[false],       |
|       range_key([t.c1], [t.__pk_increment]), range(1,MIN ; 1,MAX),                     |
|       range_cond([t.c1 = 1])                                                           |
+----------------------------------------------------------------------------------------+
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hints LEADING Hint

SELECT /*+ LEADING(table_name_list)*/ ... ;

```
EXPLAIN BASIC SELECT /*+LEADING(d c b a)*/ * FROM t1 a, t1 b, t1 c, t1 d;
+----------------------------------------------------------------------------------------+
| Query Plan                                                                             |
+----------------------------------------------------------------------------------------+
| ======================================                                                 |
| |ID|OPERATOR                          |NAME|                                          |
| --------------------------------------                                                 |
| |0 |NESTED-LOOP JOIN CARTESIAN        |    |                                          |
| |1 |├─NESTED-LOOP JOIN CARTESIAN      |    |                                          |
| |2 |│ ├─NESTED-LOOP JOIN CARTESIAN    |    |                                          |
| |3 |│ │ ├─TABLE FULL SCAN             |d   |                                          |
| |4 |│ │ └─MATERIAL                    |    |                                          |
| |5 |│ │   └─TABLE FULL SCAN           |c   |                                          |
| |6 |│ └─MATERIAL                      |    |                                          |
| |7 |│   └─TABLE FULL SCAN            |b   |                                          |
| |8 |└─MATERIAL                        |    |                                          |
| |9 |  └─TABLE FULL SCAN               |a   |                                          |
| ======================================                                                 |
+----------------------------------------------------------------------------------------+
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hints LEADING Hint

SELECT /*+ LEADING(table_name_list)*/ ... ;

```
EXPLAIN BASIC SELECT /*+LEADING((d c) (b a))*/ * FROM  t1 a, t1 b, t1 c, t1 d;
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ========================================                                                |
| |ID|OPERATOR                          |NAME|                                           |
| ----------------------------------------                                                |
| |0 |NESTED-LOOP JOIN CARTESIAN        |    |                                           |
| |1 |├─NESTED-LOOP JOIN CARTESIAN      |    |                                           |
| |2 |│ ├─TABLE FULL SCAN                |d   |                                           |
| |3 |│ └─MATERIAL                       |    |                                           |
| |4 |│   └─TABLE FULL SCAN              |c   |                                           |
| |5 |└─MATERIAL                        |    |                                           |
| |6 |  └─NESTED-LOOP JOIN CARTESIAN    |    |                                           |
| |7 |    ├─TABLE FULL SCAN             |b   |                                           |
| |8 |    └─MATERIAL                    |    |                                           |
| |9 |      └─TABLE FULL SCAN           |a   |                                           |
| ========================================                                                |
+-----------------------------------------------------------------------------------------+
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hint JOIN Hint

```
SELECT /*+ join_hint_name ( @ qb_name table_name_list) */ ... ;
```

The basic semantics is that when the right table of the join matches table_name_list, a plan is generated according to the Hint semantics. Generally, you need to use the LEADING Hint to specify the join order so that the table in table_name_list is the right table. Otherwise, the Hint will become invalid as the join order changes.

Taking nest loop join as an example, table_name_list can be in the following forms:

- Single table use_nl (t1): Nested Loop Join is used when t1 is the right table.
- Multiple single tables use_nl (t1 t2 ...): Nested Loop Join is used when t1 or t2 is the right table.
- Multiple tables use_nl ((t1 t2)): Nested Loop Join is used when t1 join t2 is the right table, ignoring the t1/t2 connection order and method.
- Multiple group tables use_nl (t1 (t2 t3) (t4 t5 t6) ...): Nested Loop Join is used when t1 / t2 join t3 / t4 join t5 join t6 is the right table.

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hint JOIN Hint

```
CREATE TABLE t0(c1 INT, c2 INT, c3 INT);

CREATE TABLE t1(c1 INT, c2 INT, c3 INT);

CREATE TABLE t2(c1 INT, c2 INT, c3 INT);
```

- If you want the join order to be t0 join t1 and the join to be a nest loop join, you should write the hint like this:

```
EXPLAIN BASIC SELECT /*+ LEADING(t0 t1) USE_NL(t1) */ * FROM t0, t1 WHERE t0.c1 = t1.c1;
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================                                                            |
| |ID|OPERATOR           |NAME|                                                           |
| ----------------------------                                                            |
| |0 |NESTED-LOOP JOIN   |    |                                                           |
| |1 |├─TABLE FULL SCAN  |t0  |                                                           |
| |2 |└─MATERIAL         |    |                                                           |
| |3 |  └─TABLE FULL SCAN|t1  |                                                           |
| ============================                                                            |
+-----------------------------------------------------------------------------------------+
```

- If you want the join order to be t0 join (t1 join t2), and you want the outermost join to be a nest loop join, you should write the hint like this:

```
EXPLAIN BASIC SELECT /*+ LEADING(t0 (t1 t2)) USE_NL((t1 t2)) */ * FROM t0, t1, t2 WHERE t0.c1 = t1.c1 AND t0.c1 = t2.c1;
+-----------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                                |
+-----------------------------------------------------------------------------------------------------------+
| ==============================                                                                            |
| |ID|OPERATOR             |NAME|                                                                           |
| ------------------------------                                                                            |
| |0 |NESTED-LOOP JOIN     |    |                                                                           |
| |1 |├─TABLE FULL SCAN    |t0  |                                                                           |
| |2 |└─MATERIAL           |    |                                                                           |
| |3 |  └─HASH JOIN        |    |                                                                           |
| |4 |    ├─TABLE FULL SCAN|t1  |                                                                           |
| |5 |    └─TABLE FULL SCAN|t2  |                                                                           |
| ==============================                                                                            |
+-----------------------------------------------------------------------------------------------------------+
```

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hint JOIN Hint

The three hints USE_NL, USE_HASH, and USE_MERGE are often used together with the LEADING hint. This is because only when the right table of the join matches table_name_list, a plan will be generated according to the hint semantics.

Suppose the user wants to intervene in the join calculation method corresponding to t1 join t2 in a SQL statement:
SELECT * FROM t1, t2 WHERE t1.c1 = t2.c1;

There are six original planned spaces:
t1 nest loop join t2
t1 hash join t2
t1 merge join t2
t2 nest loop join t1
t2 hash join t1
t2 merge join t1

If hint is added: /*+ USE_NL(t1) */, the plan space is reduced to four types:
t1 nest loop join t2
t1 hash join t2
t1 merge join t2
t2 nest loop join t1

If hint is added: /*+ LEADING(t2 t1) USE_NL(t1) */, the plan space will be determined:
t2 nest loop join t1

Managing OceanBase SQL Execution Plans

Generate a Specified Plan through Hint - Common Hint JOIN Hint

```
CREATE TABLE t0(c1 INT, c2 INT, c3 INT);

CREATE TABLE t1(c1 INT, c2 INT, c3 INT);

EXPLAIN BASIC SELECT /*+LEADING(t0 t1) USE_HASH(t1)*/ * FROM t0, t1 WHERE t0.c1 = t1.c1;
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================                                                            |
| |ID|OPERATOR         |NAME|                                                            |
| ----------------------------                                                            |
| |0 |HASH JOIN        |    |                                                            |
| |1 |├─TABLE FULL SCAN|t0  |                                                            |
| |2 |└─TABLE FULL SCAN|t1  |                                                            |
| ============================                                                            |
+-----------------------------------------------------------------------------------------+

EXPLAIN BASIC SELECT /*+LEADING(t0 t1) USE_MERGE(t1)*/ * FROM t0, t1 WHERE t0.c1 = t1.c1;
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================                                                            |
| |ID|OPERATOR           |NAME|                                                           |
| ----------------------------                                                            |
| |0 |MERGE JOIN         |    |                                                           |
| |1 |├─SORT             |    |                                                           |
| |2 |│ └─TABLE FULL SCAN|t0  |                                                           |
| |3 |└─SORT             |    |                                                           |
| |4 |  └─TABLE FULL SCAN|t1  |                                                           |
| ============================                                                            |
+-----------------------------------------------------------------------------------------+
```

OCEANBASE

FROM INTRODUCTION TO PRACTICE

Thank You!

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

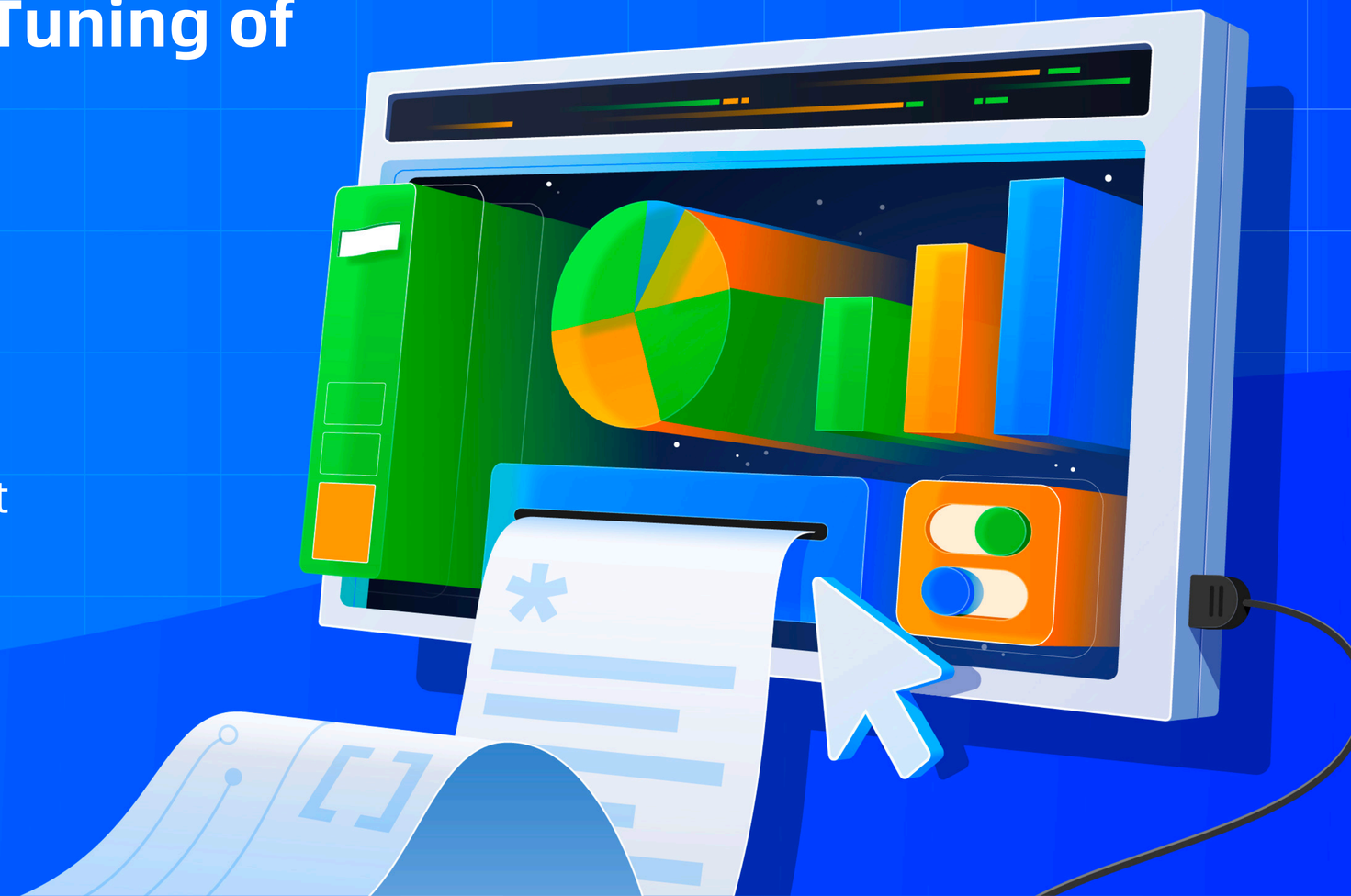
GitHub Discussions:
https://github.com/oceanbase/oceanbase/discussions

FROM INTRODUCTION TO PRACTICE

Lesson 8: Diagnosis and Tuning of OceanBase

Peng Wang

OceanBase Global Technical Evangelist



Agenda

Lesson 8.1

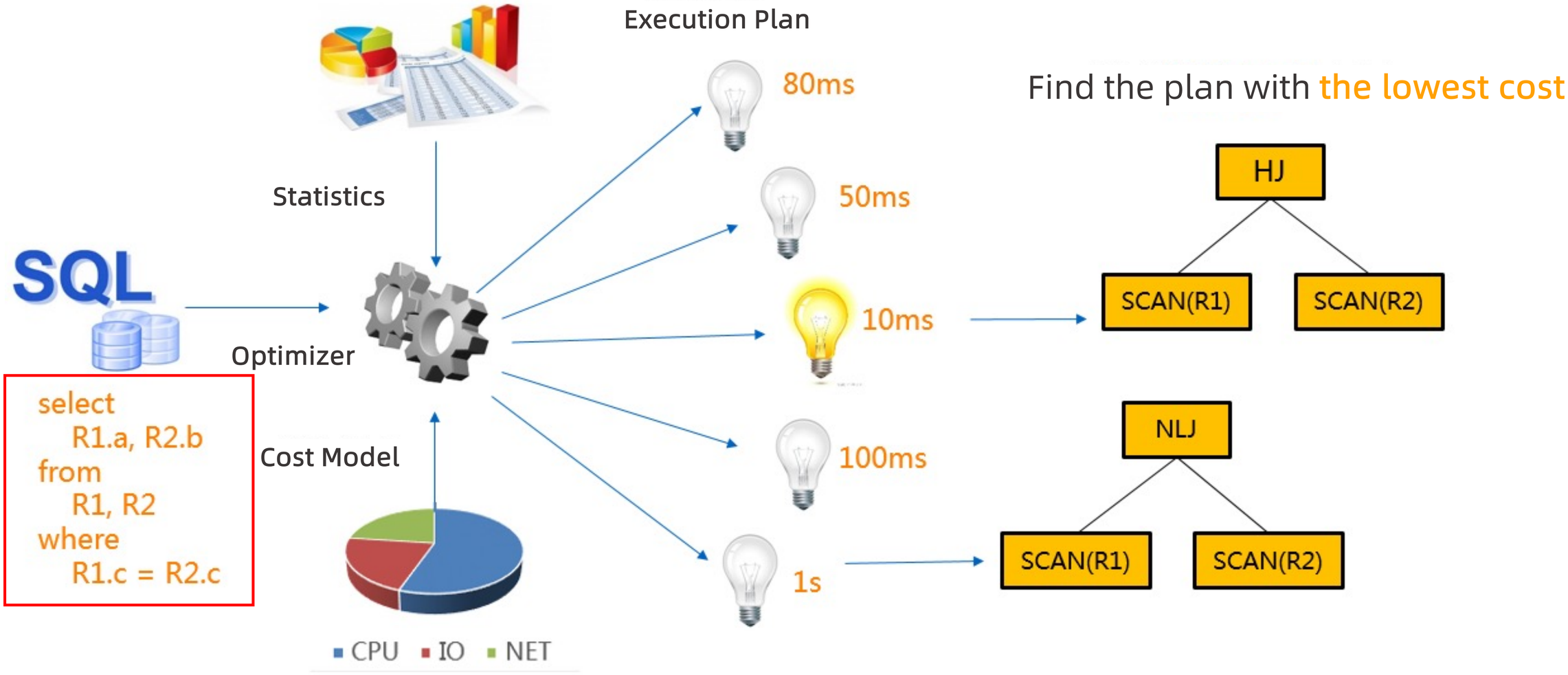
- ODP SQL Routing Principles
- Managing Database Connections
- Analyzing SQL Monitoring Views
- Read and Manage SQL Execution Plans

Lesson 8.2

- Common SQL Tuning Methods
- Troubleshooting Ideas for SQL Performance Issues

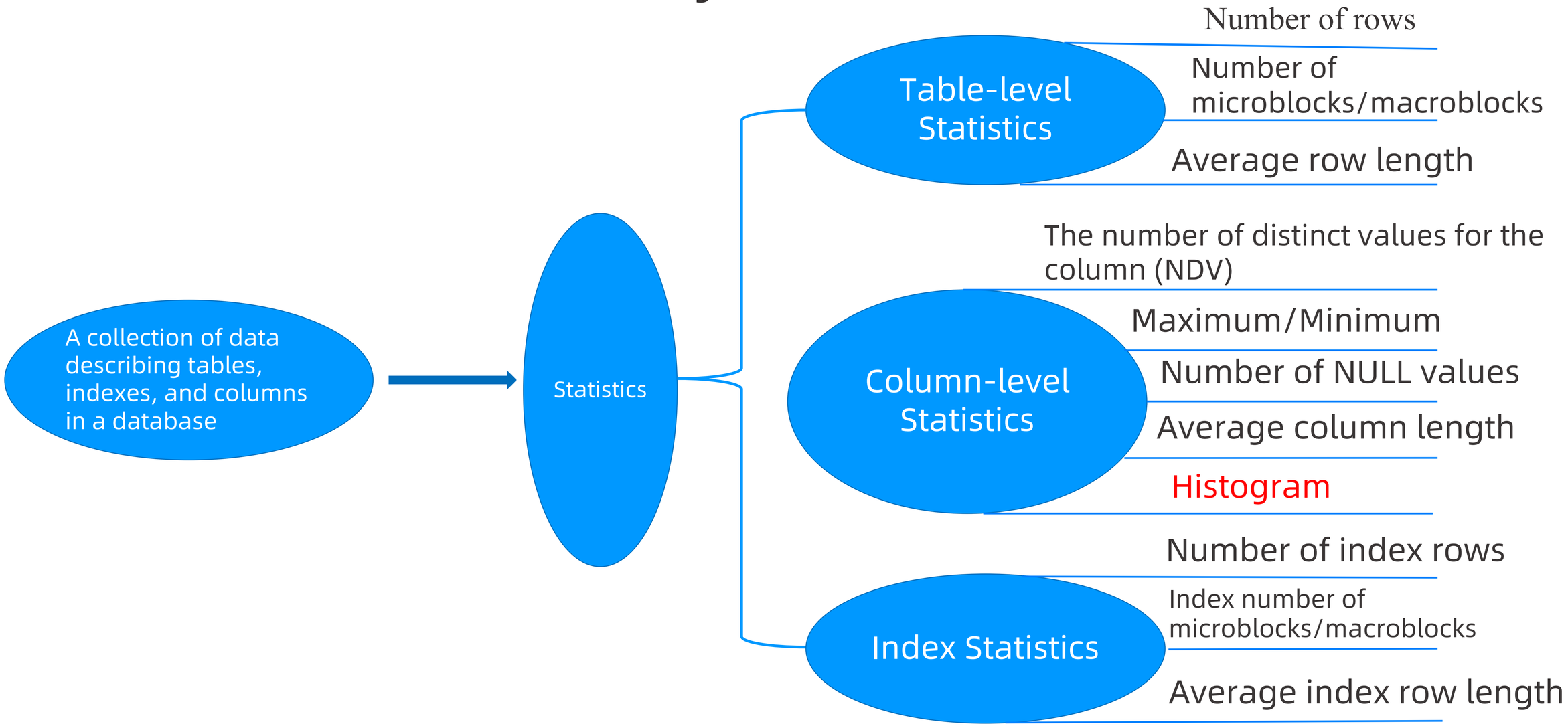
SQL Tuning Methods

Background - Statistics



SQL Tuning Methods

Background - Statistics

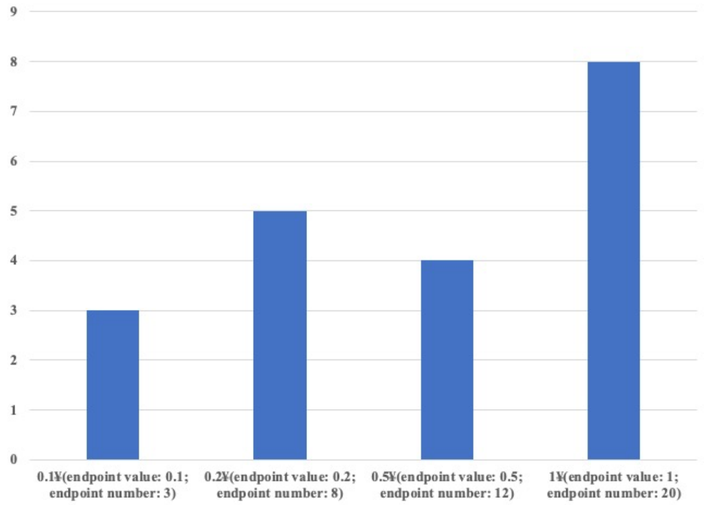


SQL Tuning Methods

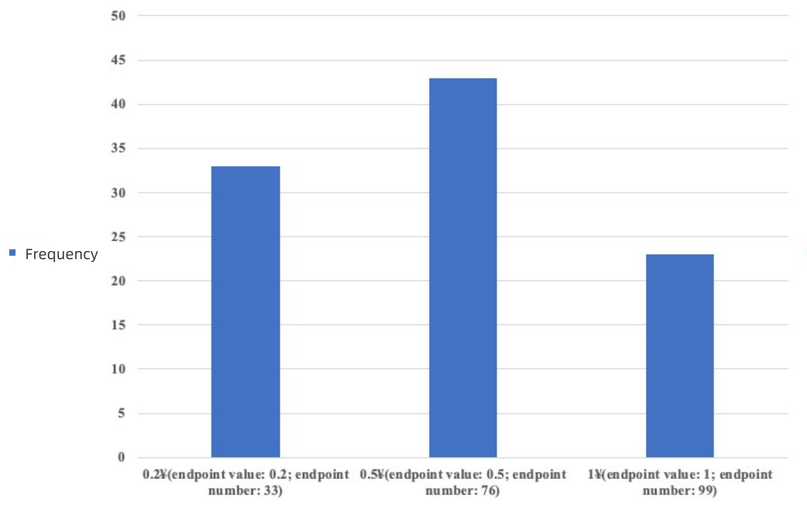
Background - Statistics

The histogram describes the data distribution, divides the different value ranges of the columns in the table into several buckets, and counts the data frequency in each bucket;
Current statistics support three types of histograms:

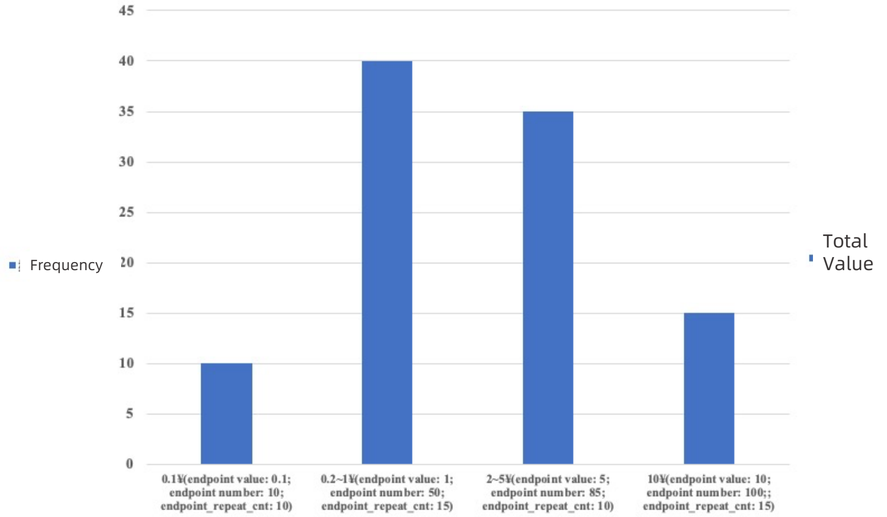
- Frequency histogram: Each distinct value corresponds to a bucket of the histogram;
- TopK histogram: When the specified number of buckets is less than NDV, the buckets with high frequency are retained;
- Mixed histogram: data is segmented according to the number of buckets, and multiple different column values are stored in one bucket, which requires sampling;



a. Frequency Histogram



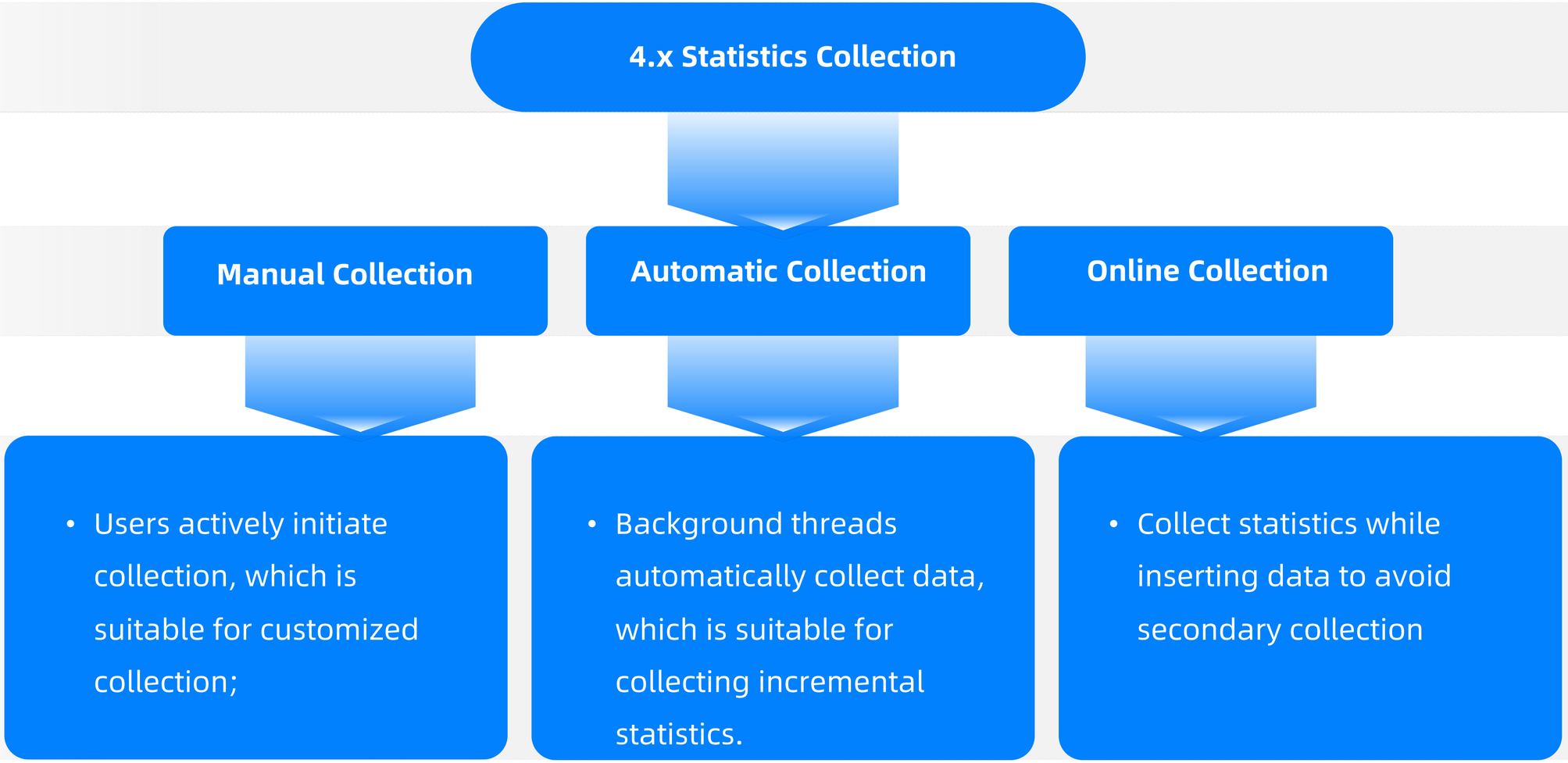
b.Topk Histogram



c. Mixed Histogram

SQL Tuning Methods

Background - Statistics



SQL Tuning Methods

Background - Manually Collecting Statistics

The two most commonly used stored procedures in the DBMS_STATS system package are:

- GATHER_TABLE_STATS, used to collect statistics of a table, syntax, and parameters are detailed on the official website
- GATHER_SCHEMA_STATS is used to collect statistics of all tables in a database. For details on syntax and parameters, see the official website

- Collect global statistics for table T1 in the TEST database. Set the number of buckets for all columns to 128.

```
call dbms_stats.gather_table_stats('TEST', 'T1', granularity=>'GLOBAL', method_opt=>'FOR ALL COLUMNS SIZE 128');
```

- Collect partition-level statistics for table T_PART1 in the TEST database, with a parallelism of 64. Only collect histograms for columns with uneven data distribution.

```
call dbms_stats.gather_table_stats('TEST', 'T_PART1', degree=>64, granularity=>'PARTITION', method_opt=>'FOR ALL COLUMNS SIZE SKEWONLY');
```

- Collect all statistics of table T_SUBPART1 in the TEST library, but only collect 50% of the data

```
call dbms_stats.gather_table_stats('TEST', 'T_SUBPART1', estimate_percent=> '50', granularity=>'ALL');
```

- Collect statistics for all tables in the TEST database, with a parallelism of 128

```
call dbms_stats.gather_schema_stats('TEST', degree=>128);
```

In addition to using the DBMS_STATS system package (recommended) to collect statistical information, you can also use the ANALYZE command in OceanBase to collect statistical information. For details on syntax and parameters, see the official website

SQL Tuning Methods

Background - Automatically Collecting Statistics

Implemented a maintenance window for statistics collection tasks based on the DBMS_SCHEDULER package:

Window Name	Start Time/Frequency	Duration Time
MONDAY_WINDOW	22:00/per week	4 hours
TUESDAY_WINDOW	22:00/per week	4 hours
WEDNESDAY_WINDOW	22:00/per week	4 hours
THURSDAY_WINDOW	22:00/per week	4 hours
FRIDAY_WINDOW	22:00/per week	4 hours
SATURDAY_WINDOW	6:00/per week	20 hours
SUNDAY_WINDOW	6:00/per week	20 hours

SQL Tuning Methods

Background - Automatically Collecting Statistics

- Query maintenance window execution information

```
select WINDOW_NAME, LAST_START_DATE, NEXT_RUN_DATE
  from OCEANBASE.DBA_SCHEDULER_WINDOWS
  where LAST_START_DATE is not null order by LAST_START_DATE;
```

```
+------------------+----------------------------+----------------------------+
| WINDOW_NAME      | LAST_START_DATE            | NEXT_RUN_DATE              |
+------------------+----------------------------+----------------------------+
| TUESDAY_WINDOW   | 2024-03-12 22:00:00.084516 | 2024-03-19 22:00:00.000000 |
| WEDNESDAY_WINDOW | 2024-03-13 22:00:00.090113 | 2024-03-20 22:00:00.000000 |
| THURSDAY_WINDOW  | 2024-03-14 22:00:00.105114 | 2024-03-21 22:00:00.000000 |
| FRIDAY_WINDOW    | 2024-03-15 22:00:00.080400 | 2024-03-22 22:00:00.000000 |
| SATURDAY_WINDOW  | 2024-03-16 06:00:00.104678 | 2024-03-23 06:00:00.000000 |
| SUNDAY_WINDOW    | 2024-03-17 06:00:00.089326 | 2024-03-24 06:00:00.000000 |
| MONDAY_WINDOW    | 2024-03-18 22:00:00.083798 | 2024-03-25 22:00:00.000000 |
+------------------+----------------------------+----------------------------+
7 rows in set (0.012 sec)
```

- Query information about all scheduled jobs

```
select JOB_NAME, REPEAT_INTERVAL, LAST_START_DATE, NEXT_RUN_DATE, MAX_RUN_DURATION
  from OCEANBASE.DBA_SCHEDULER_JOBS
  where LAST_START_DATE is not null order by LAST_START_DATE;
```

```
+--------------------------+--------------------------+----------------------------+----------------------------+------------------+
| JOB_NAME                 | REPEAT_INTERVAL          | LAST_START_DATE            | NEXT_RUN_DATE              | MAX_RUN_DURATION |
+--------------------------+--------------------------+----------------------------+----------------------------+------------------+
| TUESDAY_WINDOW           | FREQ=WEEKLY; INTERVAL=1  | 2024-03-12 22:00:00.084516 | 2024-03-19 22:00:00.000000 |            14400 |
| WEDNESDAY_WINDOW         | FREQ=WEEKLY; INTERVAL=1  | 2024-03-13 22:00:00.090113 | 2024-03-20 22:00:00.000000 |            14400 |
| THURSDAY_WINDOW          | FREQ=WEEKLY; INTERVAL=1  | 2024-03-14 22:00:00.105114 | 2024-03-21 22:00:00.000000 |            14400 |
| FRIDAY_WINDOW            | FREQ=WEEKLY; INTERVAL=1  | 2024-03-15 22:00:00.080400 | 2024-03-22 22:00:00.000000 |            14400 |
| SATURDAY_WINDOW          | FREQ=WEEKLY; INTERVAL=1  | 2024-03-16 06:00:00.104678 | 2024-03-23 06:00:00.000000 |            72000 |
| SUNDAY_WINDOW            | FREQ=WEEKLY; INTERVAL=1  | 2024-03-17 06:00:00.089326 | 2024-03-24 06:00:00.000000 |            72000 |
| MONDAY_WINDOW            | FREQ=WEEKLY; INTERVAL=1  | 2024-03-18 22:00:00.083798 | 2024-03-25 22:00:00.000000 |            14400 |
| OPT_STATS_HISTORY_MANAGER | FREQ=DAYLY; INTERVAL=1  | 2024-03-19 11:10:46.014510 | 2024-03-20 11:10:46.000000 |             NULL |
+--------------------------+--------------------------+----------------------------+----------------------------+------------------+
8 rows in set (0.017 sec)
```

SQL Tuning Methods

Background - Automatically Collecting Statistics

• Disable/enable automatic statistics task collection:
DBMS_SCHEDULER.DISABLE($window_name)
DBMS_SCHEDULER.ENABLE($window_name);

• Set the next time the automatic statistics task starts:
DBMS_SCHEDULER.SET_ATTRIBUTE($window_name, 'NEXT_DATE', $next_time);

• Set the duration of automatic statistics collection tasks:
DBMS_SCHEDULER.SET_ATTRIBUTE($window_name, 'DURATION', $duation_time);

- Disable automatic statistics collection on Monday

```
call dbms_scheduler.disable('MONDAY_WINDOW');
```

- Enable automatic statistics collection on Mondays

```
call dbms_scheduler.enable('MONDAY_WINDOW');
```

- Set the time to start collecting statistics automatically on Mondays at 20:00

```
call dbms_scheduler.set_attribute('MONDAY_WINDOW', 'NEXT_DATE', '2022-09-12 20:00:00');
```

- Set the duration of automatic statistics collection on Monday to 6 hours

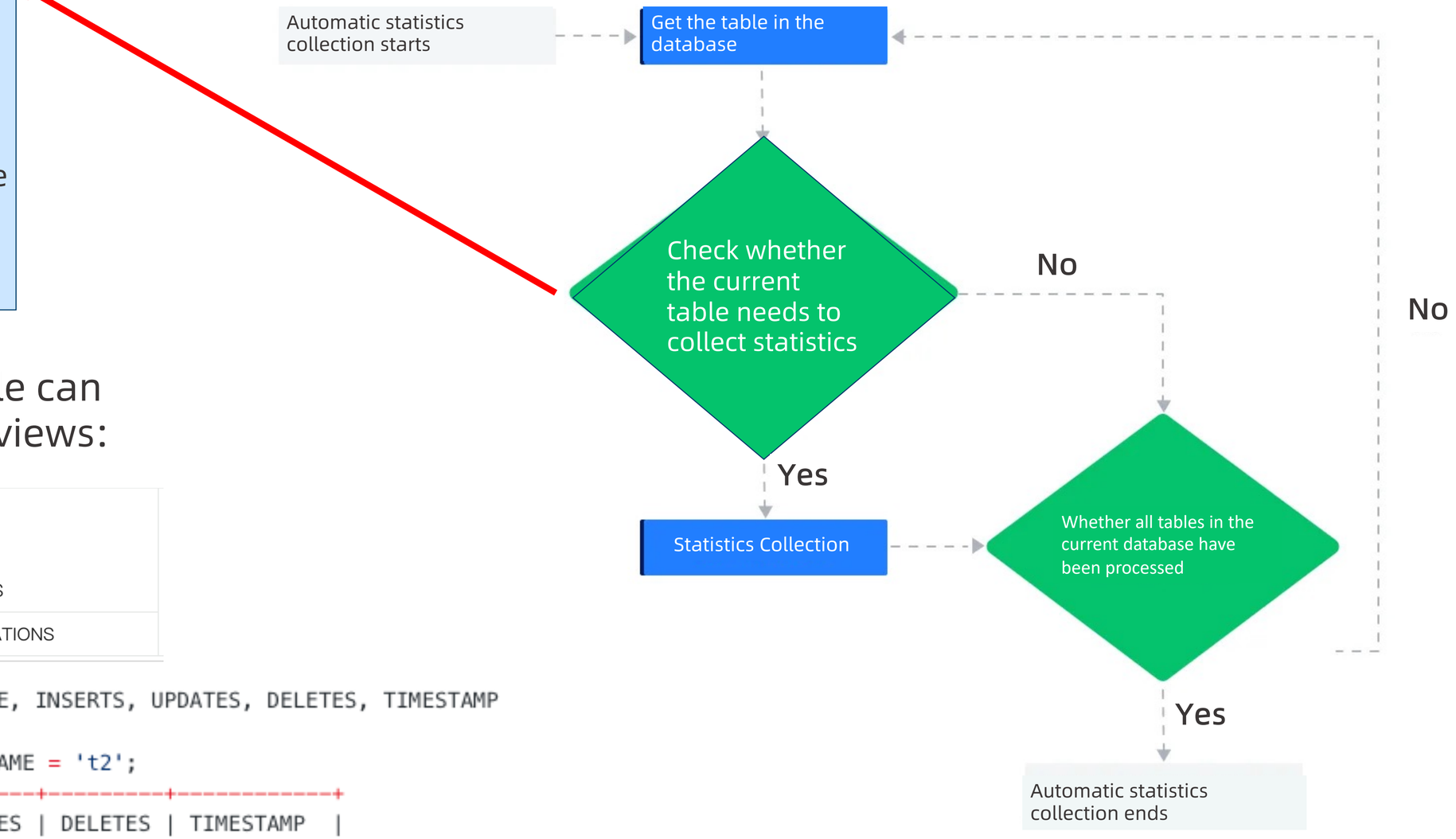
```
call dbms_scheduler.set_attribute('MONDAY_WINDOW', 'DURATION', INTERVAL '6' HOUR);
```

SQL Tuning Methods

Background - How Automatic Statistics Collection Works

Whether the ratio of additions/deletions/modifications of the table from the last collection to the current collection reaches the threshold. The default value is 10%.

If the ratio of addition/deletion/modification of some partitions of the partition table exceeds the threshold, the statistics of expired partitions will be collected again.



The DML changes of the table can be queried through related views:

Oracle	ALL_TAB_MODIFICATIONS DBA_TAB_MODIFICATIONS USER_TAB_MODIFICATIONS
Mysql	OCEANBASE.DBA_TAB_MODIFICATIONS

```
select TABLE_OWNER as DB_NAME, TABLE_NAME, INSERTS, UPDATES, DELETES, TIMESTAMP
  from OCEANBASE.DBA_TAB_MODIFICATIONS
  where TABLE_OWNER = 'test' and TABLE_NAME = 't2';
+---------+------------+---------+---------+---------+------------+
| DB_NAME | TABLE_NAME | INSERTS | UPDATES | DELETES | TIMESTAMP  |
+---------+------------+---------+---------+---------+------------+
| test    | t2         |    1000 |       0 |       0 | 2024-03-15 |
+---------+------------+---------+---------+---------+------------+
```

SQL Tuning Methods

FROM INTRODUCTION TO PRACTICE

Background Knowledge - Query Statistics

OceanBase - Oracle Mode	ALL_TAB_STATISTICS	Used to query table-level statistics
	DBA_TAB_STATISTICS	
	USER TAB STATISTICS	
	ALL IND_STATISTICS	Used to query index statistics
	DBA IND_STATISTICS	
	USER IND STATISTICS	
OceanBase - MySQL Mode	OCEANBASE.DBA_TAB_STATISTICS	Used to query index statistics
	OCEANBASE.DBA_IND_STATISTICS	Used to query table-level statistics

a. Table-level/index statistics query-related views

OceanBase - Oracle Mode	ALL_TAB_COL_STATISTICS	Used to query GLOBAL column-level statistics
	DBA_TAB_COL_STATISTICS	
	USER_TAB_COL_STATISTICS	
	ALL_PART_COL_STATISTICS	Used to query PARTITION: column-level statistics
	DBA_PART_COL_STATISTICS	
	USER_PART_COL_STATISTICS	
	ALL_SUBPART_COL_STATISTICS	Used to query SUBPARTITON: column-level statistics
	DBA_SUBPART_COL_STATISTICS	
	USER_SUBPART_COL_STATISTICS	
OceanBase - MySQL Mode	OCEANBASE.DBA_TAB_COL_STATISTICS	Used to query GLOBAL column-level statistics
	OCEANBASE.DBA_PART_COL_STATISTICS	Used to query PARTITION column-level statistics
	OCEANBASE.DBA SUBPART COL STATISTICS	Used to query SUBPARTITON column-level statistics

b. Column-level statistics query-related views

OceanBase - Oracle Mode	ALL_TAB_HISTOGRAMS	Used to query GLOBAL column-level histogram information
	DBA_TAB_HISTOGRAMS	
	USER_TAB_HISTOGRAMS	
	ALL_PART_HISTOGRAMS	Used to query PARTITION: column-level histogram information
	DBA_PART_HISTOGRAMS	
	USER_PART_HISTOGRAMS	
	ALL_SUBPART_HISTOGRAMS	Used to query SUBPARTITION column-level histogram information
	DBA_SUBPART_HISTOGRAMS	
	USER_SUBPART_HISTOGRAMS	
OceanBase - MySQL Mode	OCEANBASE.DBA_TAB_HISTOGRAMS	Used to query GLOBAL column-level histogram statistics
	OCEANBASE.DBA_PART_HISTOGRAMS	Used to query PARTITION column-level histogram information
	OCEANBASE.DBA_SUBPART_HISTOGRAMS	Used to query SUBPARTITION column-level histogram information

c. Histogram statistics query-related views

SQL Tuning Methods

Background Knowledge - Query Statistics

- Create a partition table t_part

```
create table t_part(c1 int) partition by hash(c1) partitions 4;
```

- Table-level statistics for partition table t_part. Since it has not been collected yet, the content is empty.

```
insert into t_part with recursive cte(n) as (select 1 from dual union all select n + 1 from cte where n < 1000) select 1 from cte;
```

- Query the table-level statistics of t_part. Because it has not been collected yet, the content is empty

```
select TABLE_NAME, PARTITION_NAME, PARTITION_POSITION, OBJECT_TYPE, NUM_ROWS, AVG_ROW_LEN, LAST_ANALYZED
  from OCEANBASE.DBA_TAB_STATISTICS
  where OWNER = 'test' and TABLE_NAME = 't_part';
+------------+----------------+--------------------+-------------+----------+-------------+---------------+
| TABLE_NAME | PARTITION_NAME | PARTITION_POSITION | OBJECT_TYPE | NUM_ROWS | AVG_ROW_LEN | LAST_ANALYZED |
+------------+----------------+--------------------+-------------+----------+-------------+---------------+
| t_part     | p3             |                  4 | PARTITION   |     NULL |        NULL | NULL          |
| t_part     | p2             |                  3 | PARTITION   |     NULL |        NULL | NULL          |
| t_part     | p1             |                  2 | PARTITION   |     NULL |        NULL | NULL          |
| t_part     | p0             |                  1 | PARTITION   |     NULL |        NULL | NULL          |
| t_part     | NULL           |               NULL | TABLE       |     NULL |        NULL | NULL          |
+------------+----------------+--------------------+-------------+----------+-------------+---------------+
5 rows in set (0.238 sec)

explain select * from t_part where c1 > 2;
```

- Manually collect statistics

```
analyze table t_part COMPUTE STATISTICS for all columns size 128;
```

- Query table-level statistics of t_part

```
select TABLE_NAME, PARTITION_NAME, PARTITION_POSITION, OBJECT_TYPE, NUM_ROWS, AVG_ROW_LEN, LAST_ANALYZED
  from OCEANBASE.DBA_TAB_STATISTICS
  where OWNER = 'test' and TABLE_NAME = 't_part';
+------------+----------------+--------------------+-------------+----------+-------------+----------------------------+
| TABLE_NAME | PARTITION_NAME | PARTITION_POSITION | OBJECT_TYPE | NUM_ROWS | AVG_ROW_LEN | LAST_ANALYZED              |
+------------+----------------+--------------------+-------------+----------+-------------+----------------------------+
| t_part     | NULL           |               NULL | TABLE       |     1000 |          20 | 2024-03-22 14:11:03.188965 |
| t_part     | p0             |                  1 | PARTITION   |        0 |           0 | 2024-03-22 14:11:03.188965 |
| t_part     | p1             |                  2 | PARTITION   |     1000 |          20 | 2024-03-22 14:11:03.188965 |
| t_part     | p2             |                  3 | PARTITION   |        0 |           0 | 2024-03-22 14:11:03.188965 |
| t_part     | p3             |                  4 | PARTITION   |        0 |           0 | 2024-03-22 14:11:03.188965 |
+------------+----------------+--------------------+-------------+----------+-------------+----------------------------+
5 rows in set (0.032 sec)
```

SQL Tuning Methods

Background Knowledge - Query Statistics

- Because the c1 column in the table only has data with a value of 1, the estimated row for the plan with c1 > 1 is very small, which is in line with expectations.

```
explain select * from t_part where c1 > 1;
```

```
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================================================                            |
| |ID|OPERATOR                  |NAME    |EST.ROWS|EST.TIME(us)|                        |
| ------------------------------------------------------------                            |
| |0 |PX COORDINATOR            |        |1       |64          |                        |
| |1 |└─EXCHANGE OUT DISTR      |:EX10000|1       |64          |                        |
| |2 |  └─PX PARTITION ITERATOR |        |1       |64          |                        |
| |3 |    └─TABLE FULL SCAN     |t_part  |1       |64          |                        |
| ============================================================                            |
| Outputs & filters:                                                                      |
| -------------------------------------                                                   |
|   0 - output([INTERNAL_FUNCTION(t_part.c1)]), filter(nil), rowset=16                    |
|   1 - output([INTERNAL_FUNCTION(t_part.c1)]), filter(nil), rowset=16                    |
|       dop=1                                                                             |
|   2 - output([t_part.c1]), filter(nil), rowset=16                                       |
|       force partition granule                                                           |
|   3 - output([t_part.c1]), filter([t_part.c1 > 1]), rowset=16                           |
|       access([t_part.c1]), partitions(p[0-3])                                           |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false],      |
|       range_key([t_part.__pk_increment]), range(MIN ; MAX)always true                   |
+-----------------------------------------------------------------------------------------+
```

SQL Tuning Methods

FROM INTRODUCTION TO PRACTICE

Background Knowledge - Query Statistics

- Then insert 1000 rows of data into the partition table t_part, and the values of the data are all 99

```
insert into t_part with recursive cte(n)
  as (select 1 from dual union all select n + 1 from cte where n < 1000) select 99 from cte;
```

- After 1000 rows of data with a value of 99 appear in the c1 column of the table, the optimizer has yet to collect the data in time, so the estimate for the c1>1 plan is still small, which is not in line with expectations.
- If the optimizer does not have the latest statistics, it may generate a suboptimal plan. If you see a problematic row, it is recommended that statistics be collected manually.
- For example, in the figure below, operator 3 has 1000 output rows after filter([t_part.c1 > 1]), which is more than the 1 row shown by EST.ROWS. At this time, you can manually collect statistics.

```
explain select * from t_part where c1 > 1;
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================================================                            |
| |ID|OPERATOR                  |NAME    |EST.ROWS|EST.TIME(us)|                        |
| ------------------------------------------------------------                            |
| |0 |PX COORDINATOR            |        |1       |64          |                        |
| |1 |└─EXCHANGE OUT DISTR      |:EX10000|1       |64          |                        |
| |2 |  └─PX PARTITION ITERATOR |        |1       |64          |                        |
| |3 |    └─TABLE FULL SCAN     |t_part  |1       |64          |                        |
| ============================================================                            |
| Outputs & filters:                                                                      |
| -------------------------------------                                                   |
|   0 - output([INTERNAL_FUNCTION(t_part.c1)]), filter(nil), rowset=16                    |
|   1 - output([INTERNAL_FUNCTION(t_part.c1)]), filter(nil), rowset=16                    |
|       dop=1                                                                             |
|   2 - output([t_part.c1]), filter(nil), rowset=16                                       |
|       force partition granule                                                           |
|   3 - output([t_part.c1]), filter([t_part.c1 > 1]), rowset=16                           |
|       access([t_part.c1]), partitions(p[0-3])                                           |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false],      |
|       range_key([t_part.__pk_increment]), range(MIN ; MAX)always true                   |
+-----------------------------------------------------------------------------------------+
```

SQL Tuning Methods

- Manually collect statistics

```
analyze table t_part COMPUTE STATISTICS for all columns size 128;
```

- After manually collecting statistics, you can see that the estimated number of rows is in line with expectations.

```
explain select * from t_part where c1 > 1;
```

```
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================================================                            |
| |ID|OPERATOR                  |NAME    |EST.ROWS|EST.TIME(us)|                        |
| ------------------------------------------------------------                            |
| |0 |PX COORDINATOR            |        |1000    |707         |                        |
| |1 |└─EXCHANGE OUT DISTR      |:EX10000|1000    |515         |                        |
| |2 |  └─PX PARTITION ITERATOR |        |1000    |87          |                        |
| |3 |    └─TABLE FULL SCAN     |t_part  |1000    |87          |                        |
| ============================================================                            |
| Outputs & filters:                                                                      |
| -------------------------------------                                                   |
|   0 - output([INTERNAL_FUNCTION(t_part.c1)]), filter(nil), rowset=256                   |
|   1 - output([INTERNAL_FUNCTION(t_part.c1)]), filter(nil), rowset=256                   |
|       dop=1                                                                             |
|   2 - output([t_part.c1]), filter(nil), rowset=256                                      |
|       force partition granule                                                           |
|   3 - output([t_part.c1]), filter([t_part.c1 > 1]), rowset=256                          |
|       access([t_part.c1]), partitions(p[0-3])                                           |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false],       |
|       range_key([t_part.__pk_increment]), range(MIN ; MAX)always true                   |
+-----------------------------------------------------------------------------------------+
```

SQL Tuning Methods

Background Knowledge - Indexes In OB

Q: In addition to the index key, OceanBase's index also contains the primary key of the main table. Why?

We can do a simple experiment and create an index called idx_b

```
create table test(a int primary key, b int, c int, key idx_b(b));
```

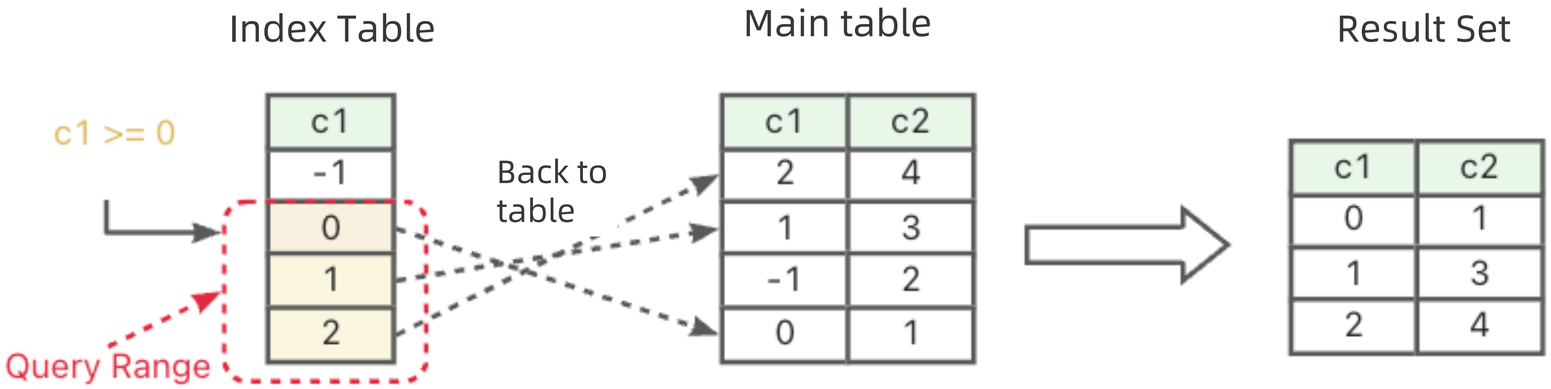
```
select
  column_id,
  column_name,
  rowkey_position,
  index_position
from
  oceanbase.__all_column
where
  table_id = (
    select
      table_id
    from
      oceanbase.__all_table
    where
      data_table_id = (
        select
          table_id
        from
          oceanbase.__all_table
        where
          table_name = 'test'
      )
  );
+-----------+-------------+-----------------+----------------+
| column_id | column_name | rowkey_position | index_position |
+-----------+-------------+-----------------+----------------+
|        16 | a           |               2 |              0 |
|        17 | b           |               1 |              1 |
+-----------+-------------+-----------------+----------------+
```

SQL Tuning Methods

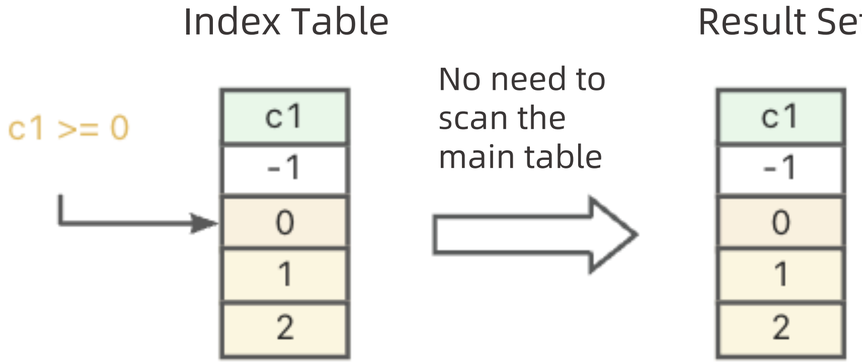
Background Knowledge - Indexes in OB

```
obclient [test]> create table t1(c1 int, c2 int, index idx(c1));
Query OK, 0 rows affected (0.226 sec)

obclient [test]> select
    ->   column_id,
    ->   column_name,
    ->   rowkey_position,
    ->   index_position
    -> from
    ->   oceanbase.__all_column
    -> where
    ->   table_id = (
    ->     select
    ->       table_id
    ->     from
    ->       oceanbase.__all_table
    ->     where
    ->       data_table_id = (
    ->         select
    ->           table_id
    ->         from
    ->           oceanbase.__all_table
    ->         where
    ->           table_name = 't1'
    ->       )
    ->   );
+-----------+---------------+-----------------+----------------+
| column_id | column_name   | rowkey_position | index_position |
+-----------+---------------+-----------------+----------------+
|        16 | c1            |               1 |              1 |
|         1 | __pk_increment|               2 |              0 |
+-----------+---------------+-----------------+----------------+
2 rows in set (0.002 sec)
```

```
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;
```



SQL Tuning Methods

Background Knowledge - The Role of Indexes

- **Reduce the number of rows read:** You can quickly locate data based on the conditions of the index column to reduce the amount of data scanned

- **Eliminate sorting:** Index columns are ordered, and this feature can be used to eliminate some sorting operations

- **Save I/O:** The columns on the index are generally fewer than those on the main table. If the filtering conditions are good or the number of columns to be queried is small, you can scan less column data that does not need to be queried, saving system I/O resources.

SQL Tuning Methods

The Role of Indexes - Quickly Locate Data

```
create table test(a int primary key, b int, c int, d int, key idx_b_c(b, c));
```

- Let's insert some data to verify whether the data on index idx_b_c is in order of b, c, and a.

```
insert into test values(1, 2, 3, 4);

insert into test values(2, 3, 4, 5);

insert into test values(5, 1, 2, 3);

insert into test values(4, 1, 3, 3);

insert into test values(3, 1, 3, 3);
```

- The default scanning method for the entire table is to scan the primary table, and the rows you see are sorted by the primary key column a.

```
select * from test;
```

```
+---+---+---+---+
| a | b | c | d |
+---+---+---+---+
| 1 | 2 | 3 | 4 |
| 2 | 3 | 4 | 5 |
| 3 | 1 | 2 | 3 |
| 4 | 1 | 3 | 3 |
| 5 | 1 | 2 | 3 |
+---+---+---+---+
5 rows in set (0.001 sec)
```

- By using hint to force the index idx_b_c, the row order you see becomes sorted by index columns b and c.
- This is the actual data order on the index. Knowing this will help you understand the subsequent content.

```
select /*+ index(test idx_b_c) */ * from test;
```

```
+---+---+---+---+
| a | b | c | d |
+---+---+---+---+
| 3 | 1 | 2 | 3 |
| 5 | 1 | 2 | 3 |
| 4 | 1 | 3 | 3 |
| 1 | 2 | 3 | 4 |
| 2 | 3 | 4 | 5 |
+---+---+---+---+
5 rows in set (0.005 sec)
```

SQL Tuning Methods

The Role of Indexes - Quickly Locate Data

create table test(a int primary key, b int, c int, d int, key idx_b_c(b, c));

```
explain select /*+ index(test idx_b_c) */ * from test where b = 1;
+-----------------------------------------------------------------------------------+
| Query Plan                                                                        |
+-----------------------------------------------------------------------------------+
| ============================================================                      |
| |ID|OPERATOR        |NAME         |EST.ROWS|EST.TIME(us)|                      |
| ------------------------------------------------------------                      |
| |0 |TABLE RANGE SCAN|test(idx_b_c)|1       |5           |                      |
| ============================================================                      |
| Outputs & filters:                                                                |
| -------------------------------------                                             |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256     |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)              |
|       is_index_back=true, is_global_index=false,                                  |
|       range_key([test.b], [test.c], [test.a]), range(1,MIN,MIN ; 1,MAX,MAX),      |
|       range_cond([test.b = 1])                                                    |
+-----------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Quickly Locate Data

create table test(a int primary key, b int, c int, d int, key idx_b_c(b, c));

```
explain select /*+index(test idx_b_c)*/ * from test where b > 1;
+-----------------------------------------------------------------------------------------------------+
| Query Plan                                                                                          |
+-----------------------------------------------------------------------------------------------------+
| ============================================================                                        |
| |ID|OPERATOR        |NAME         |EST.ROWS|EST.TIME(us)|                                        |
| ------------------------------------------------------------                                        |
| |0 |TABLE RANGE SCAN|test(idx_b_c)|1       |5           |                                        |
| ============================================================                                        |
| Outputs & filters:                                                                                  |
| -------------------------------------                                                               |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256                       |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)                                |
|       is_index_back=true, is_global_index=false,                                                    |
|       range_key([test.b], [test.c], [test.a]), range(1,MAX,MAX ; MAX,MAX,MAX),                      |
|       range_cond([test.b > 1])                                                                      |
+-----------------------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Quickly Locate Data

create table test(a int primary key, b int, c int, d int, key idx_b_c(b, c));

```
explain select/*+index(test idx_b_c)*/ * from test where b = 1 and c > 1;
+-----------------------------------------------------------------------------+
| Query Plan                                                                  |
+-----------------------------------------------------------------------------+
| ============================================================                |
| |ID|OPERATOR        |NAME         |EST.ROWS|EST.TIME(us)|                   |
| ------------------------------------------------------------                |
| |0 |TABLE RANGE SCAN|test(idx_b_c)|1       |5           |                   |
| ============================================================                |
| Outputs & filters:                                                          |
| -------------------------------------                                       |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256 |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)        |
|       is_index_back=true, is_global_index=false,                            |
|       range_key([test.b], [test.c], [test.a]), range(1,1,MAX ; 1,MAX,MAX),  |
|       range_cond([test.b = 1], [test.c > 1])                                |
+-----------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Quickly Locate Data

create table test(a int primary key, b int, c int, d int, key idx_b_c(b, c));

```
explain select /*+ index(test idx_b_c) */ * from test where b > 1 and c > 1;
+-----------------------------------------------------------------------------------------------------+
| Query Plan                                                                                          |
+-----------------------------------------------------------------------------------------------------+
| ==========================================================                                          |
| |ID|OPERATOR        |NAME         |EST.ROWS|EST.TIME(us)|                                          |
| ----------------------------------------------------------                                          |
| |0 |TABLE RANGE SCAN|test(idx_b_c)|1       |3           |                                          |
| ==========================================================                                          |
| Outputs & filters:                                                                                  |
| ------------------------------------                                                                |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter([test.c > 1]), rowset=256              |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)                                |
|       is_index_back=true, is_global_index=false, filter_before_indexback[true],                     |
|       range_key([test.b], [test.c], [test.a]), range(1,MAX,MAX ; MAX,MAX,MAX),                      |
|       range_cond([test.b > 1])                                                                      |
+-----------------------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Eliminating The Overhead Of Sorting

create table test(a int primary key, b int, c int, d int, key idx_b_c_d(b, c, d));

```
explain select /*+ index(test idx_b_c_d) */ * from test where b = 1 order by c;
+-----------------------------------------------------------------------------------------------+
| Query Plan                                                                                    |
+-----------------------------------------------------------------------------------------------+
| ============================================================                                  |
| |ID|OPERATOR        |NAME            |EST.ROWS|EST.TIME(us)|                                  |
| ------------------------------------------------------------                                  |
| |0 |TABLE RANGE SCAN|test(idx_b_c_d)|1       |2           |                                  |
| ============================================================                                  |
| Outputs & filters:                                                                            |
| -------------------------------------                                                         |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256                 |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)                          |
|       is_index_back=false, is_global_index=false,                                             |
|       range_key([test.b], [test.c], [test.d], [test.a]), range(1,MIN,MIN,MIN ; 1,MAX,MAX,MAX), |
|       range_cond([test.b = 1])                                                                |
+-----------------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Eliminating The Overhead Of Sorting

create table test(a int primary key, b int, c int, d int, key idx_b_c_d(b, c, d));

```
explain select /*+ index(test idx_b_c_d) */ * from test where b = 1 or b = 2 order by c;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| ============================================================                                                                                                                            |
| |ID|OPERATOR          |NAME           |EST.ROWS|EST.TIME(us)|                                                                                                                                            |
| ------------------------------------------------------------                                                                                                                            |
| |0 |SORT              |               |1       |2           |                                                                                                                                            |
| |1 |└─TABLE RANGE SCAN|test(idx_b_c_d)|1       |2           |                                                                                                                                            |
| ============================================================                                                                                                                            |
| Outputs & filters:                                                                                                                                                                      |
| -------------------------------------                                                                                                                                                   |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256                                                                                                           |
|       sort_keys([test.c, ASC])                                                                                                                                                          |
|   1 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256                                                                                                           |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)                                                                                                                    |
|       is_index_back=false, is_global_index=false,                                                                                                                                       |
|       range_key([test.b], [test.c], [test.d], [test.a]), range(1,MIN,MIN,MIN ; 1,MAX,MAX,MAX), (2,MIN,MIN,MIN ; 2,MAX,MAX,MAX),                                                        |
|       range_cond([test.b = 1 OR test.b = 2])                                                                                                                                            |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Eliminating The Overhead Of Sorting

create table test(a int primary key, b int, c int, d int, key idx_b_c_d(b, c, d));

explain select /*+ index(test idx_b_c_d) */ * from test where b = 1 and c = 2 order by c;

```
explain select /*+ index(test idx_b_c_d) */ * from test where b = 1 and c = 2 order by c;
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================================================                            |
| |ID|OPERATOR        |NAME            |EST.ROWS|EST.TIME(us)|                           |
| ------------------------------------------------------------                            |
| |0 |TABLE RANGE SCAN|test(idx_b_c_d)|1       |2           |                           |
| ============================================================                            |
| Outputs & filters:                                                                      |
| -------------------------------------                                                   |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter(nil), rowset=256           |
|       access([test.a], [test.b], [test.c], [test.d]), partitions(p0)                    |
|       is_index_back=false, is_global_index=false,                                       |
|       range_key([test.b], [test.c], [test.d], [test.a]), range(1,2,MIN,MIN ; 1,2,MAX,MAX) |
|       range_cond([test.b = 1], [test.c = 2])                                            |
+-----------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Eliminating The Overhead Of Sorting

create table test(a int primary key, b int, c int, d int, key idx_b_c_d(b, c, d));

explain select /*+ index(test idx_b_c_d) */ * from test where c = 1 order by b, d;

```
explain select /*+ index(test idx_b_c_d) */ * from test where c = 1 order by b, d;
+-----------------------------------------------------------------------------------------------------------------------------+
| Query Plan                                                                                                                  |
+-----------------------------------------------------------------------------------------------------------------------------+
| ============================================================                                                                |
| |ID|OPERATOR       |NAME           |EST.ROWS|EST.TIME(us)|                                                                |
| ------------------------------------------------------------                                                                |
| |0 |TABLE FULL SCAN|test(idx_b_c_d)|1       |2           |                                                                |
| ============================================================                                                                |
| Outputs & filters:                                                                                                          |
| -------------------------------------                                                                                       |
|   0 - output([test.a], [test.b], [test.c], [test.d]), filter([test.c = 1]), rowset=256                                      |
|       access([test.a], [test.c], [test.b], [test.d]), partitions(p0)                                                        |
|       is_index_back=false, is_global_index=false, filter_before_indexback[false],                                          |
|       range_key([test.b], [test.c], [test.d], [test.a]), range(MIN,MIN,MIN,MIN ; MAX,MAX,MAX,MAX)always true               |
+-----------------------------------------------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Scanning Less Data

```
create table test(a int, b int, c int, d int, key idx_b(b));

explain select b from test;
```

```
+-----------------------------------------------------------------------------------------+
| Query Plan                                                                              |
+-----------------------------------------------------------------------------------------+
| ============================================                                            |
| |ID|OPERATOR       |NAME       |EST.ROWS|EST.TIME(us)|                                  |
| --------------------------------------------                                            |
| |0 |TABLE FULL SCAN|test(idx_b)|1       |2           |                                  |
| ============================================                                            |
| Outputs & filters:                                                                      |
| -------------------------------------                                                   |
|   0 - output([test.b]), filter(nil), rowset=256                                         |
|       access([test.b]), partitions(p0)                                                  |
|       is_index_back=false, is_global_index=false,                                       |
|       range_key([test.b], [test.__pk_increment]), range(MIN,MIN ; MAX,MAX)always true   |
+-----------------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Scanning Less Data

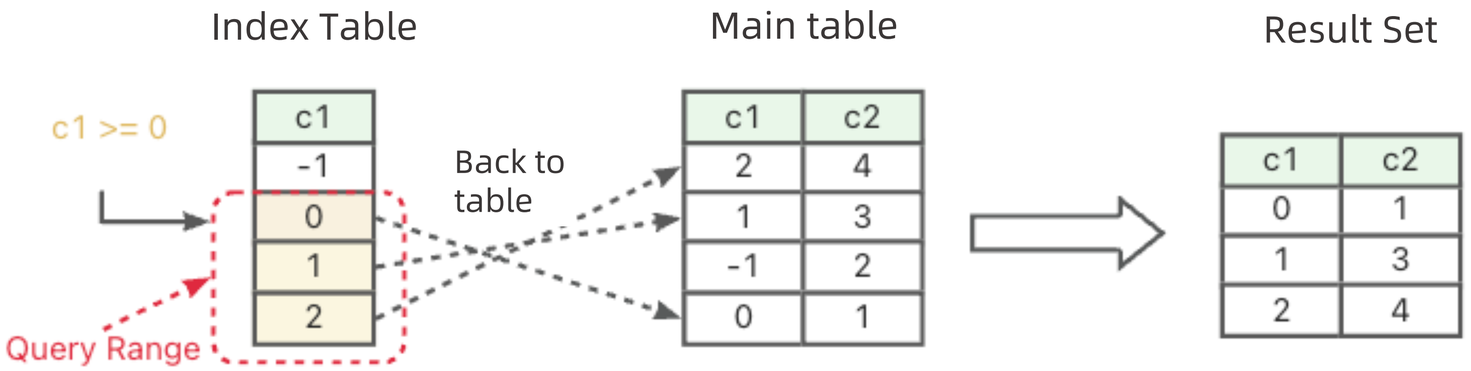
```
create table test(a int, b int, c int, d int, key idx_b(b));

explain select count(*) from test;
```

```
+-----------------------------------------------------------------------------------------------+
| Query Plan                                                                                    |
+-----------------------------------------------------------------------------------------------+
| ==========================================================                                    |
| |ID|OPERATOR         |NAME       |EST.ROWS|EST.TIME(us)|                                    |
| ----------------------------------------------------------                                    |
| |0 |SCALAR GROUP BY  |           |1       |2           |                                    |
| |1 |└─TABLE FULL SCAN|test(idx_b)|1       |2           |                                    |
| ==========================================================                                    |
| Outputs & filters:                                                                            |
| -------------------------------------                                                         |
|   0 - output([T_FUN_COUNT_SUM(T_FUN_COUNT(*))]), filter(nil), rowset=256                      |
|       group(nil), agg_func([T_FUN_COUNT_SUM(T_FUN_COUNT(*))])                                 |
|   1 - output([T_FUN_COUNT(*)]), filter(nil), rowset=256                                       |
|       access(nil), partitions(p0)                                                             |
|       is_index_back=false, is_global_index=false,                                             |
|       range_key([test.b], [test.__pk_increment]), range(MIN,MIN ; MAX,MAX)always true,        |
|       pushdown_aggregation([T_FUN_COUNT(*)])                                                  |
+-----------------------------------------------------------------------------------------------+
```

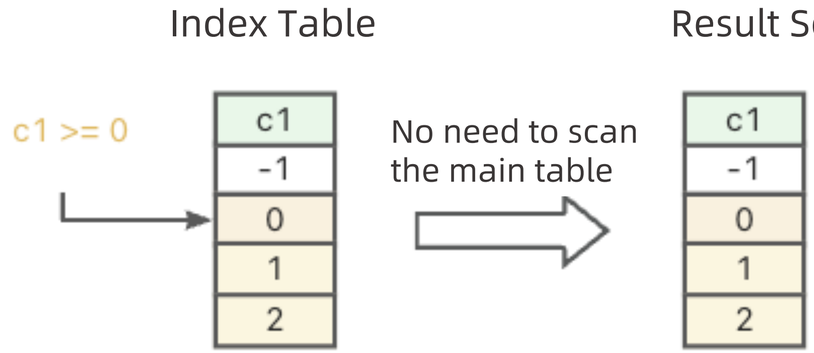
SQL Tuning Methods

The Role of Indexes - Scanning Less Data

```
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;
```



SQL Tuning Methods

The Role of Indexes - Scanning Less Data

```
create table test(a int, b int, c int, d int, key idx_b(b));

explain select a, b from test;
+-----------------------------------------------------------------------------+
| Query Plan                                                                  |
+-----------------------------------------------------------------------------+
| ============================================                                |
| |ID|OPERATOR       |NAME|EST.ROWS|EST.TIME(us)|                            |
| --------------------------------------------                                |
| |0 |TABLE FULL SCAN|test|1       |2           |                            |
| ============================================                                |
| Outputs & filters:                                                          |
| -------------------------------------                                       |
|   0 - output([test.a], [test.b]), filter(nil), rowset=256                   |
|       access([test.a], [test.b]), partitions(p0)                            |
|       is_index_back=false, is_global_index=false,                           |
|       range_key([test.__pk_increment]), range(MIN ; MAX)always true         |
+-----------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - Scanning Less Data

```
explain select /*+ index(test idx_b) */ a, b from test;
+-----------------------------------------------------------------------------------+
| Query Plan                                                                        |
+-----------------------------------------------------------------------------------+
| ============================================                                      |
| |ID|OPERATOR       |NAME        |EST.ROWS|EST.TIME(us)|                          |
| --------------------------------------------                                      |
| |0 |TABLE FULL SCAN|test(idx_b)|1       |5           |                          |
| ============================================                                      |
| Outputs & filters:                                                                |
| -------------------------------------                                             |
|   0 - output([test.a], [test.b]), filter(nil), rowset=256                         |
|       access([test.__pk_increment], [test.a], [test.b]), partitions(p0)           |
|       is_index_back=true, is_global_index=false,                                  |
|       range_key([test.b], [test.__pk_increment]), range(MIN,MIN ; MAX,MAX)always true |
+-----------------------------------------------------------------------------------+
```

SQL Tuning Methods

The Role of Indexes - How to Measure the Time It Takes to Index

The time spent on indexing consists of two parts:

1. Time to scan the index (determined by the number of rows to scan)
2. Time to return the index to the table (determined by the number of rows to return)

```
create table test(a int primary key, b int, c int, d int, e int, key idx_b_e_c(b, e, c));
```

Assume that this table has 10,000 rows of data. The time to scan the index and main table is 1 ms for 1,000 rows, and the time for indexing back to the table is 1 ms for 100 rows (roughly ten times the time).

Filtering with the condition b = 1 and c = 1 will return 1000 rows of data;

```
select /*+index(test idx_b_c_d)*/ a, b, c from test where b = 1;      =>     1ms
select /*+index(test primary)*/ a, b, c from test where b = 1;          =>     1ms * 10 = 10ms
```

SQL Tuning Methods

The Role of Indexes - How to Measure The Time It Takes to Index

```
select /*+index(test idx_b_c_d)*/ * from test where b = 1 and c = 1 ;          =>   1 ms + 10 ms = 11 ms。

select /*+index(test primary)*/ * from test where b = 1 and c = 1;             =>   1ms * 10 = 10ms
```

Q: How to get information like "Filtering with b = 1 and c = 1 will return 1000 rows of data”?
Very simple! Just execute a SQL statement and check the count:
select count(*) from test where b = 1 and c = 1;

```
+------------------+
|         count(*) |
+------------------+
|             1000 |
+------------------+
```

SQL Tuning Methods

Index Tuning Summary

The strategy for creating indexes can be roughly summarized in the following three sentences:

- Put the columns with equal value conditions at the front of the index, and put the columns with range conditions at the back of the index.
- When there are range conditions on multiple columns, put the columns with strong filtering properties at the front.
- Common covering indexes can effectively avoid table backlogs

For example, there are three filtering conditions in a SQL statement:

- a = 1, b > 0, and c between 1 and 12.
- b > 0 can filter out 30% of the data,
- c between 1 and 12 can filter out 90% of the data

Q: So how should we build an index?

Build an index like: Idx (a, c, b)

“b”: To eliminate the overhead of returning to the table when selecting *

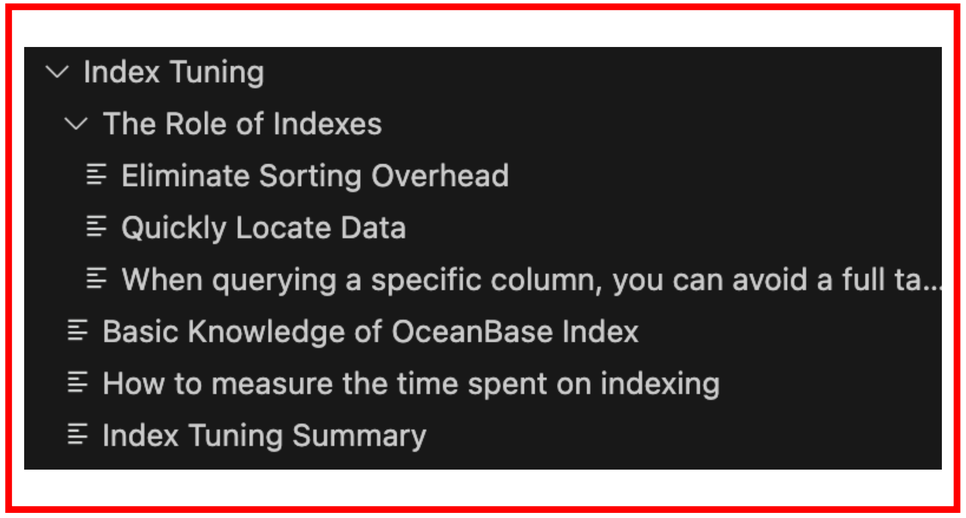
SQL Tuning Methods

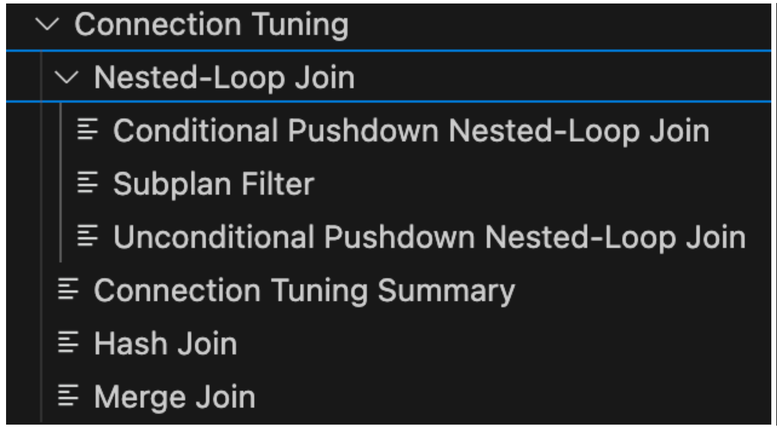
Other SQL tuning knowledge

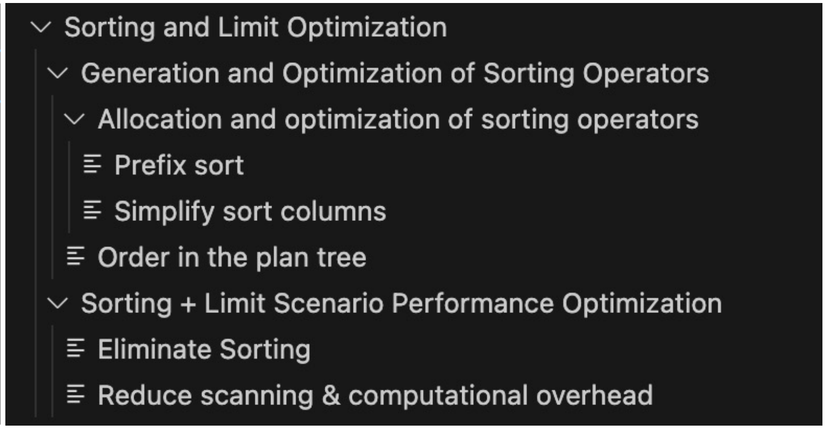
More than 90% of the SQL tuning problems reported by support engineers are caused by not knowing how to create appropriate indexes:

- Is it necessary to create an index?
- On which columns should the index be created?
- How should the order of index columns be arranged?

.....







Agenda

Lesson 8.1

- ODP SQL Routing Principles
- Managing Database Connections
- Analyzing SQL Monitoring Views
- Read and Manage SQL Execution Plans

Lesson 8.2

- Common SQL Tuning Methods
- Troubleshooting Ideas for SQL Performance Issues

Troubleshooting SQL Performance Issues

Troubleshooting Steps

When users encounter performance problems caused by SQL, they can generally troubleshoot by following the steps below:

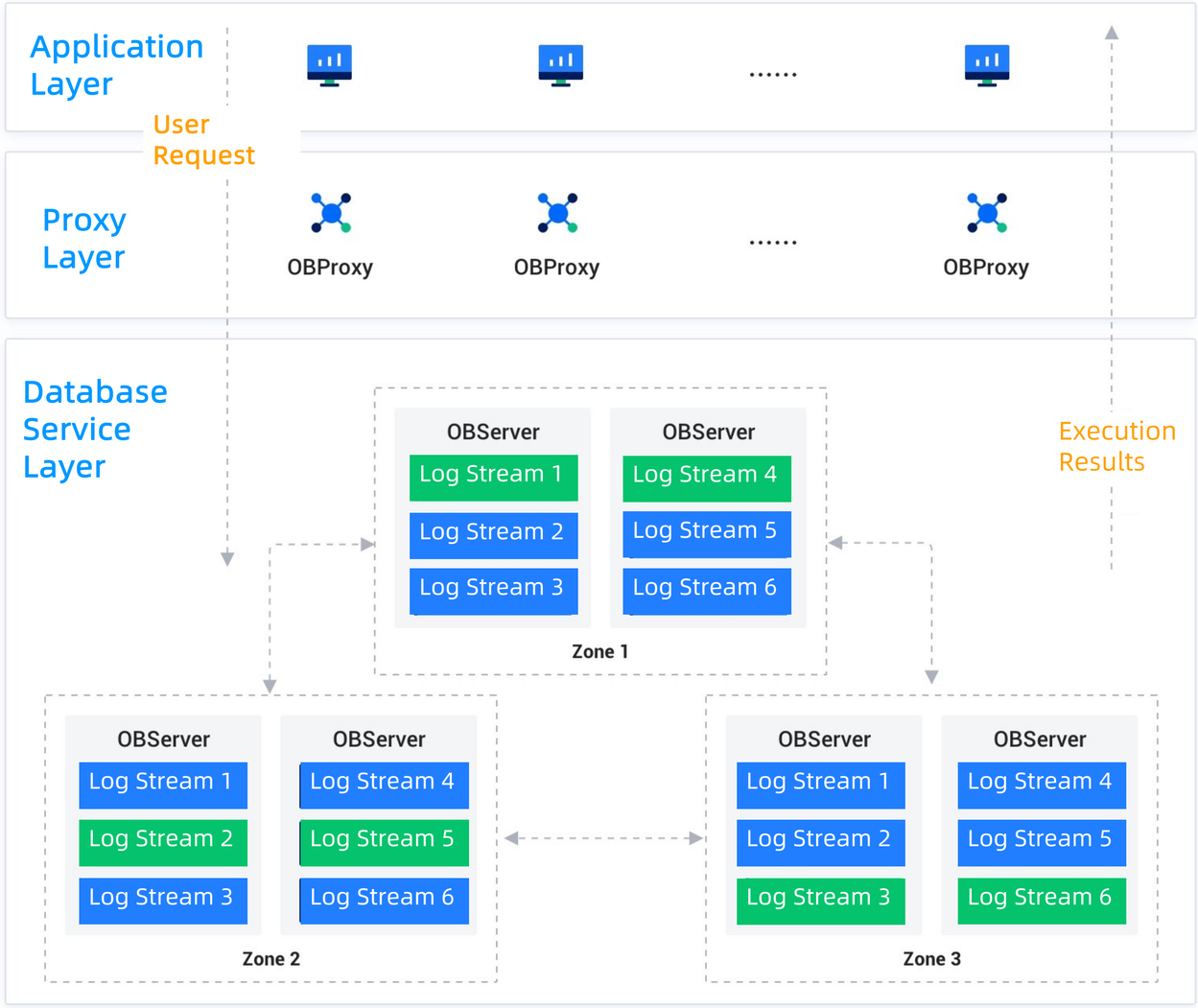
1. Through full-link tracking, confirm the time consumption ratio of each stage and identify which stage takes the longest time.
2. If the previous step shows that the slowness lies in the observer module, you can use oceanbase.gv$ob_sql_audit to analyze which stage in the observer takes a long time.
3. If the previous step takes a long time in the execution phase, first determine whether there are problems such as buffer table, large and small accounts, hard parsing, etc. based on the above content
4. If none of the above problems exist, you need to use the execution plan displayed by explain extended to analyze whether there is a huge gap between the optimizer's estimated number of rows and the actual number of rows. If there is a significant gap, you need to manually collect statistics. Otherwise, you need to further consider whether you need to create a more appropriate index, adjust the plan shape through hints, adjust the degree of parallelism through hints, etc.

Troubleshooting SQL Performance Issues

SQL Performance Problem Analysis Tool - Full-link Tracking

Command: show trace

The data link of the OceanBase database is APPServer <-> OBProxy <-> OBServer



FROM INTRODUCTION TO PRACTICE

Troubleshooting SQL Performance Issues

SQL Performance Problem Analysis Tool - Full-link Tracking

```
create table t1(c1 int);

insert into t1 values(123);
```

- Enable the show trace function of the full-link diagnosis in the current session

```
SET ob_enable_show_trace = ON;
```

```
SELECT c1 FROM t1 LIMIT 2;
+------+
| c1   |
+------+
|  123 |
+------+
1 row in set (0.23 sec)
```

```
SHOW TRACE;
```

Operation	StartTime	ElapseTime
ob_proxy	2024-03-20 15:07:46.419433	191.999 ms
├── ob_proxy_partition_location_lookup	2024-03-20 15:07:46.419494	181.839 ms
├── ob_proxy_server_process_req	2024-03-20 15:07:46.601697	9.138 ms
└── com_query_process	2024-03-20 15:07:46.601920	8.824 ms
└── mpquery_single_stmt	2024-03-20 15:07:46.601940	8.765 ms
├── sql_compile	2024-03-20 15:07:46.601984	7.666 ms
│ ├── pc_get_plan	2024-03-20 15:07:46.602051	0.029 ms
│ └── hard_parse	2024-03-20 15:07:46.602195	7.423 ms
│ ├── parse	2024-03-20 15:07:46.602201	0.137 ms
│ ├── resolve	2024-03-20 15:07:46.602393	0.555 ms
│ ├── rewrite	2024-03-20 15:07:46.603104	1.055 ms
│ ├── optimize	2024-03-20 15:07:46.604194	4.298 ms
│ │ ├── inner_execute_read	2024-03-20 15:07:46.605959	0.825 ms
│ │ │ ├── sql_compile	2024-03-20 15:07:46.606078	0.321 ms
│ │ │ │ └── pc_get_plan	2024-03-20 15:07:46.606124	0.147 ms
│ │ │ ├── open	2024-03-20 15:07:46.606418	0.129 ms
│ │ │ └── do_local_das_task	2024-03-20 15:07:46.606606	0.095 ms
│ │ └── close	2024-03-20 15:07:46.606813	0.240 ms
│ │ ├── close_das_task	2024-03-20 15:07:46.606879	0.022 ms
│ │ └── end_transaction	2024-03-20 15:07:46.607009	0.023 ms
│ ├── code_generate	2024-03-20 15:07:46.608527	0.374 ms
│ └── pc_add_plan	2024-03-20 15:07:46.609375	0.207 ms
└── sql_execute	2024-03-20 15:07:46.609677	0.832 ms
├── open	2024-03-20 15:07:46.609684	0.156 ms
├── response_result	2024-03-20 15:07:46.609875	0.327 ms
│ └── do_local_das_task	2024-03-20 15:07:46.609905	0.136 ms
└── close	2024-03-20 15:07:46.610221	0.225 ms
├── close_das_task	2024-03-20 15:07:46.610229	0.029 ms
└── end_transaction	2024-03-20 15:07:46.610410	0.019 ms

Troubleshooting SQL Performance Issues

SQL Performance Problem Analysis Tool - Full-link Tracking

```
create table t1(c1 int);

insert into t1 values(123);
```

- Enable the show trace function of the full-link diagnosis in the current session

```
SET ob_enable_show_trace = ON;
```

```
SELECT c1 FROM t1 LIMIT 2;
+------+
| c1   |
+------+
|  123 |
+------+
1 row in set (0.23 sec)
```

```
SELECT c1 FROM t1 LIMIT 2;
+------+
| c1   |
+------+
|  123 |
+------+
1 row in set (0.01 sec)

SHOW TRACE;
+-----------------------------------------------+----------------------------+-------------+
| Operation                                     | StartTime                  | ElapseTime  |
+-----------------------------------------------+----------------------------+-------------+
| ob_proxy                                      | 2024-03-20 15:34:14.879559 | 7.390 ms    |
| ├── ob_proxy_partition_location_lookup        | 2024-03-20 15:34:14.879652 | 4.691 ms    |
| ├── ob_proxy_server_process_req               | 2024-03-20 15:34:14.884785 | 1.514 ms    |
| └── com_query_process                         | 2024-03-20 15:34:14.884943 | 1.237 ms    |
|     └── mpquery_single_stmt                   | 2024-03-20 15:34:14.884959 | 1.207 ms    |
|         ├── sql_compile                       | 2024-03-20 15:34:14.884997 | 0.279 ms    |
|         │   └── pc_get_plan                   | 2024-03-20 15:34:14.885042 | 0.071 ms    |
|         └── sql_execute                       | 2024-03-20 15:34:14.885300 | 0.809 ms    |
|             ├── open                          | 2024-03-20 15:34:14.885310 | 0.139 ms    |
|             ├── response_result               | 2024-03-20 15:34:14.885513 | 0.314 ms    |
|             │   └── do_local_das_task         | 2024-03-20 15:34:14.885548 | 0.114 ms    |
|             └── close                         | 2024-03-20 15:34:14.885847 | 0.190 ms    |
|                 ├── close_das_task            | 2024-03-20 15:34:14.885856 | 0.030 ms    |
|                 └── end_transaction           | 2024-03-20 15:34:14.885997 | 0.019 ms    |
+-----------------------------------------------+----------------------------+-------------+
```

Troubleshooting SQL Performance Issues

SQL Performance Problem Analysis Tool - Full-link Tracking

```
create table t1(c1 int);

insert into t1 values(123);
```

- Enable the show trace function of the full-link diagnosis in the current session

SET ob_enable_show_trace = **ON**;

```
SELECT c1 FROM t1 LIMIT 2;
+------+
| c1   |
+------+
|  123 |
+------+
1 row in set (0.23 sec)
```

```
SELECT c1 FROM t1 LIMIT 2;
+------+
| c1   |
+------+
|  123 |
+------+
1 row in set (0.05 sec)

show trace;
+----------------------------------------+----------------------------+-----------+
| Operation                              | StartTime                  | ElapseTime |
+----------------------------------------+----------------------------+-----------+
| com_query_process                      | 2024-03-20 15:54:38.772699 | 1.746 ms  |
| └── mpquery_single_stmt                | 2024-03-20 15:54:38.772771 | 1.647 ms  |
|     ├── sql_compile                    | 2024-03-20 15:54:38.772835 | 0.356 ms  |
|     │   └── pc_get_plan                | 2024-03-20 15:54:38.772900 | 0.143 ms  |
|     └── sql_execute                    | 2024-03-20 15:54:38.773209 | 1.052 ms  |
|         ├── open                       | 2024-03-20 15:54:38.773232 | 0.150 ms  |
|         ├── response_result            | 2024-03-20 15:54:38.773413 | 0.421 ms  |
|         │   └── do_local_das_task      | 2024-03-20 15:54:38.773479 | 0.192 ms  |
|         └── close                      | 2024-03-20 15:54:38.773857 | 0.379 ms  |
|             ├── close_das_task         | 2024-03-20 15:54:38.773913 | 0.069 ms  |
|             └── end_transaction        | 2024-03-20 15:54:38.774139 | 0.058 ms  |
+----------------------------------------+----------------------------+-----------+
11 rows in set (0.07 sec)
```

Troubleshooting SQL Performance Issues

SQL Performance Problem Analysis Tool - sql audit

The general idea of analyzing abnormal SQL in the oceanbase.gv$ob_sql_audit view is:

- Use the GET_PLAN_TIME field to check the time to get the execution plan. If it is very long, it is usually accompanied by IS_HIT_PLAN = 0, indicating that the plan cache was not hit, resulting in a complete hard parsing process.
- Through the two fields SSSTORE_READ_ROW_COUNT and MEMSTORE_READ_ROW_COUNT, you can analyze whether the SQL performance problem may be caused by the Buffer table problem.
- Use the QUEUE_TIME field to check whether the queue time is very long. If QUEUE_TIME is large, it means that the tenant's worker thread has a waiting problem. Further analysis is needed to determine whether the waiting problem is reasonable.
- Check the RETRY_CNT field to see if the number of retries is high. If so, further check for lock conflicts or master switching.
- Through the value of the EXECUTE_TIME field, you can combine the GV$OB_PLAN_CACHE_PLAN_EXPLAIN dictionary view to analyze whether the execution plan is reasonable. If it is not reasonable, you need to further consider whether you need to create a suitable index, whether you need to adjust the plan shape through hints, etc.

Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Buffer Table Issues

The problem with the Buffer table is that OceanBase uses a storage engine based on LSM-Tree:
Under the LSM-Tree architecture, deleted data is marked for deletion and will not take effect before merging.

- Create a table for testing the Buffer table

```
create table t1(c1 int);
```

- Insert 1000 rows of data

```
insert into t1 with recursive cte(n) as (select 1 from dual union all select n + 1 from cte where n < 1000) select n from cte;
```

- Delete alternate rows, delete 500 rows in total

```
delete from t1 where c1 % 2 = 0;
```

- Pay attention to the information of physical_range_rows and logical_range_rows

```
explain extended_noaddr select * from t1;
+----------------------------------------------------------------------+
| Query Plan                                                           |
+----------------------------------------------------------------------+
| ============================================                         |
| |ID|OPERATOR       |NAME|EST.ROWS|EST.TIME(us)|                     |
| --------------------------------------------                         |
| |0 |TABLE FULL SCAN|t1  |501     |38          |                     |
| ============================================                         |
| Outputs & filters:                                                   |
| -------------------------------------                                |
|   0 - output([t1.c1]), filter(nil), rowset=256                       |
|       access([t1.c1]), partitions(p0)                                |
|       is_index_back=false, is_global_index=false,                    |
|       range_key([t1.__pk_increment]), range(MIN ; MAX)always true    |
| ......                                                               |
| Optimization Info:                                                   |
| -------------------------------------                                |
|   t1:                                                                |
|       table_rows:501                                                 |
|       physical_range_rows:1000                                       |
|       logical_range_rows:500                                         |
|       index_back_rows:0                                              |
|       output_rows:501                                                |
|       table_dop:1                                                    |
|       dop_method:Table DOP                                           |
|       avaiable_index_name:[t1]                                       |
|       stats version:0                                                |
|       dynamic sampling level:0                                       |
|   Plan Type:                                                         |
|       LOCAL                                                          |
|   Note:                                                              |
|       Degree of Parallelisim is 1 because of table property          |
+----------------------------------------------------------------------+
```

- Manually trigger compaction, please refer to https://en.oceanbase.com/docs/common-oceanbase-database-10000000001717304

```
ALTER SYSTEM MAJOR FREEZE;
```

- View compaction progress, please refer to https://en.oceanbase.com/docs/common-oceanbase-database-10000000001717301

```
SELECT START_TIME, LAST_FINISH_TIME, STATUS FROM oceanbase.DBA_OB_MAJOR_COMPACTION;
+----------------------------+----------------------------+--------+
| START_TIME                 | LAST_FINISH_TIME           | STATUS |
+----------------------------+----------------------------+--------+
| 2024-03-20 17:54:18.610008 | 2024-03-20 17:54:50.738156 | IDLE   |
+----------------------------+----------------------------+--------+
1 row in set (0.019 sec)
```

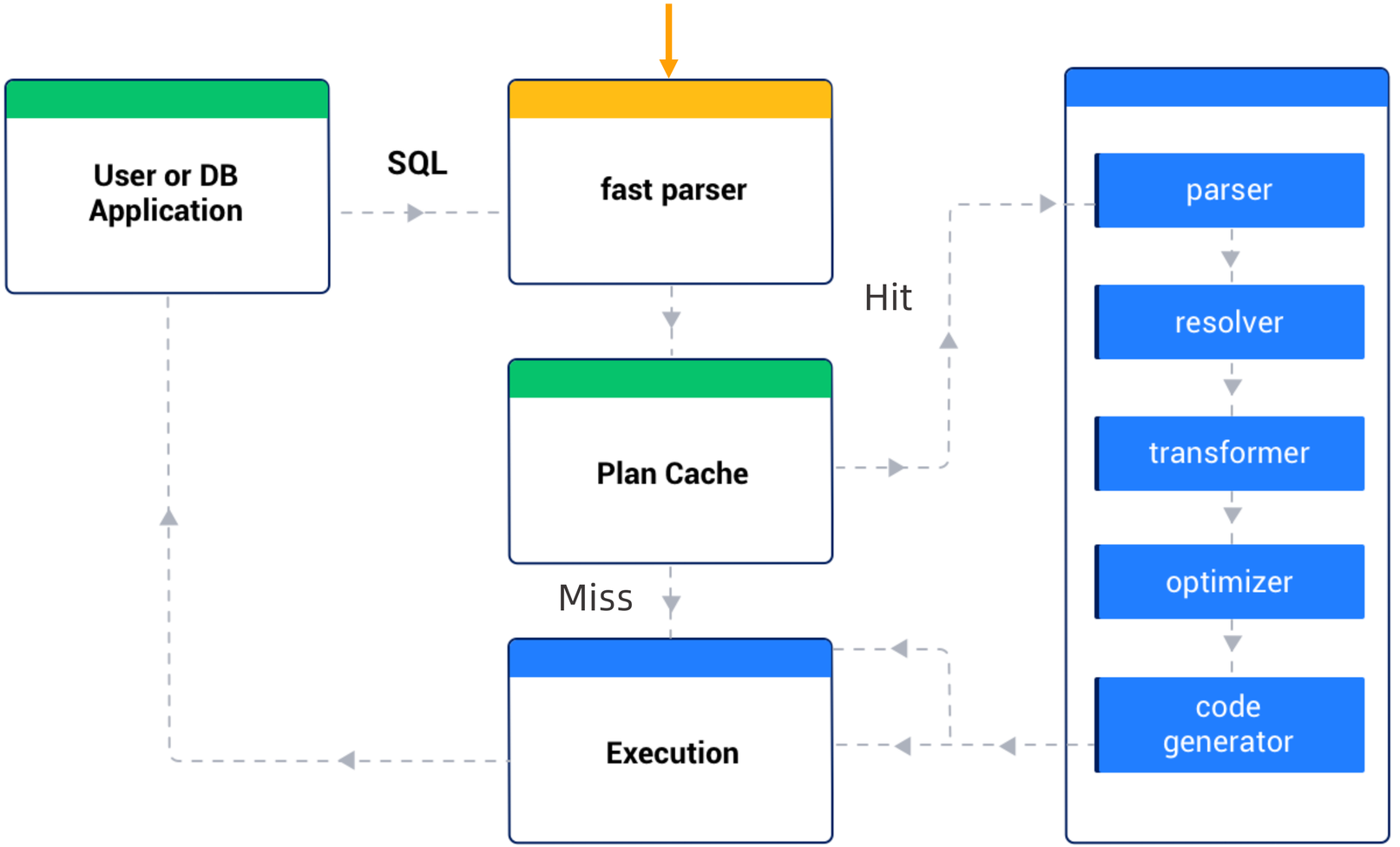
- Pay attention to the information of physical_range_rows and logical_range_rows after the compaction

```
explain extended_noaddr select * from t1;
+-----------------------------------------------------------------------+
| Query Plan                                                            |
+-----------------------------------------------------------------------+
| ============================================                          |
| |ID|OPERATOR       |NAME|EST.ROWS|EST.TIME(us)|                      |
| --------------------------------------------                          |
| |0 |TABLE FULL SCAN|t1  |501     |21          |                      |
| ============================================                          |
| Outputs & filters:                                                    |
| -------------------------------------                                 |
|   0 - output([t1.c1]), filter(nil), rowset=256                        |
|       access([t1.c1]), partitions(p0)                                 |
|       is_index_back=false, is_global_index=false,                     |
|       range_key([t1.__pk_increment]), range(MIN ; MAX)always true     |
| ......                                                                |
| Optimization Info:                                                    |
| -------------------------------------                                 |
|   t1:                                                                 |
|       table_rows:501                                                  |
|       physical_range_rows:500                                         |
|       logical_range_rows:500                                          |
|       index_back_rows:0                                               |
|       output_rows:501                                                 |
|       table_dop:1                                                     |
|       dop_method:Table DOP                                            |
|       avaiable_index_name:[t1]                                        |
|       stats version:0                                                 |
|       dynamic sampling level:0                                        |
|   Plan Type:                                                          |
|       LOCAL                                                           |
|   Note:                                                               |
|       Degree of Parallelisim is 1 because of table property           |
+-----------------------------------------------------------------------+
```

Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Bad Cases of Plan Cache (Big and Small Account Issues)

Fast parser performs a fast parameterization on the SQL text. The function of fast parameterization is to replace the constant parameters in the SQL text with wildcards ?
SELECT * FROM t1 WHERE c1 = 1 will be replaced by SELECT * FROM t1 WHERE c1 = ?



Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Bad Cases of Plan Cache (Big and Small Account Issues)

- Create a table t1

```
create table t1 (c1 int, c2 int, key idx1(c1), key idx2(c2));
```

- Insert 1000 rows of data. The c1 column has only two values, of which the value 0 accounts for 0.1% of all data and the value 1 accounts for 99.9% of all data. The c2 column has 10 values from 1 to 10, each accounting for 10%.

```
insert into t1 values(0, 0);
insert into t1 with recursive cte(n)
  as (select 1 from dual union all select n + 1 from cte where n < 999) select 1, mod(n, 10) + 1 from cte;
```

- Manually collect statistics

```
analyze table t1 COMPUTE STATISTICS for all columns size 128;
```

- The structure of this parameterized SQL statement is: select * from t1 where c1 = ? and c2 = ?

```
select * from t1 where c1 = 0 and c2 = 0;
```

- Query the plans cached in the plan cache
- Because the filtering condition of the above SQL is c1 = 0, the filtering is very good, so you can see in OUTLINE_DATA that this plan uses the index idx1

```
SELECT SQL_ID, PLAN_ID, STATEMENT, OUTLINE_DATA, PLAN_ID, LAST_ACTIVE_TIME, QUERY_SQL
  FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT
  WHERE QUERY_SQL LIKE '%select * from t1 where c1 = %'\G
*************************** 1. row ***************************
          SQL_ID: F296DCC7D661BF78D15FD5E4A753B53B
         PLAN_ID: 52941
       STATEMENT: select * from t1 where c1 = ? and c2 = ?
    OUTLINE_DATA: /*+BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "test"."t1"@"SEL$1" "idx1") OPTIMIZER_FEATURES_ENABLE('4.0.0.0') END_OUTLINE_DATA*/
         PLAN_ID: 52941
LAST_ACTIVE_TIME: 2024-03-19 17:11:18.822475
       QUERY_SQL: select * from t1 where c1 = 0 and c2 = 0
```

Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Bad Cases of Plan Cache (Big and Small Account Issues)

- Execute the following SQL, the filtering condition is c1 = 1, the filtering performance is very poor
- The result of this parameterized SQL statement is also: select * from t1 where c1 = ? and c2 = ? , so the plan just generated will be reused.

```
select * from t1 where c1 = 1 and c2 = 1;
```

- Note that the value of LAST_ACTIVE_TIME has changed, indicating that this plan has been reused.
- The above SQL with filter conditions c1 = 1 and c2 = 1 reuses the plan of the SQL with filter conditions c1 = 0 and c2 = 0 in the plan cache.

```
SELECT SQL_ID, PLAN_ID, STATEMENT, OUTLINE_DATA, PLAN_ID, LAST_ACTIVE_TIME, QUERY_SQL
  FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT
  WHERE QUERY_SQL LIKE '%select * from t1 where c1 = %'\G
*************************** 1. row ***************************
          SQL_ID: F296DCC7D661BF78D15FD5E4A753B53B
         PLAN_ID: 52941
       STATEMENT: select * from t1 where c1 = ? and c2 = ?
    OUTLINE_DATA: /*+BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "test"."t1"@"SEL$1" "idx1") OPTIMIZER_FEATURES_ENABLE('4.0.0.0') END_OUTLINE_DATA*/
         PLAN_ID: 52941
LAST_ACTIVE_TIME: 2024-03-19 17:13:25.877066
       QUERY_SQL: select * from t1 where c1 = 0 and c2 = 0
```

- When the filtering condition is c1 = 1 and c2 = 1, the index idx2 should be used, because the filtering performance of idx2 is much better than that of idx1.
- However, due to the plan cache, the actual execution plan above selects a suboptimal index.

Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Bad Cases of Plan Cache (Big and Small Account Issues)

Workaround:

- The plan is controlled through the Hint and Outline just introduced, which will not be repeated here.
- Clear the specific plan cache directly through the command, for example:
  ALTER SYSTEM FLUSH PLAN CACHE
  sql_id='F296DCC7D661BF78D15FD5E4A753B53B' databases='test' GLOBAL;

  alter system flush plan cache [tenant_list] [global];
- By changing the parameterized result, it is impossible to reuse the plan in the plan cache (non-standard approach)

Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Bad Cases of Plan Cache (Big and Small Account Issues)

- The method is: to add an extra space in the middle of the SQL to change c2 = 1 to c2 = 1
- This changes the parameterized things. See the STATEMENT field in the following SQL for details.
- Note: Spaces need to be added in the middle of the SQL, not at the beginning or end

```
select * from t1 where c1 = 1 and c2 =  1;
```

- A new plan is generated, and from OUTLINE_DATA, we can see that the index in the plan has become the more optimal idx2

```
SELECT SQL_ID, PLAN_ID, STATEMENT, OUTLINE_DATA, PLAN_ID, LAST_ACTIVE_TIME, QUERY_SQL
  FROM oceanbase.GV$OB_PLAN_CACHE_PLAN_STAT
  WHERE QUERY_SQL LIKE '%select * from t1 where c1 = %'\G
*************************** 1. row ***************************
          SQL_ID: F296DCC7D661BF78D15FD5E4A753B53B
         PLAN_ID: 52941
       STATEMENT: select * from t1 where c1 = ? and c2 = ?
    OUTLINE_DATA: /*+BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "test"."t1"@"SEL$1" "idx1") OPTIMIZER_FEATURES_ENABLE('4.0.0.0') END_OUTLINE_DATA*/
         PLAN_ID: 52941
LAST_ACTIVE_TIME: 2024-03-19 17:13:25.877066
       QUERY_SQL: select * from t1 where c1 = 0 and c2 = 0
*************************** 2. row ***************************
          SQL_ID: 3DC824F8228724AE4B1435111273F59C
         PLAN_ID: 52949
       STATEMENT: select * from t1 where c1 = ? and c2 =  ?
    OUTLINE_DATA: /*+BEGIN_OUTLINE_DATA INDEX(@"SEL$1" "test"."t1"@"SEL$1" "idx2") OPTIMIZER_FEATURES_ENABLE('4.0.0.0') END_OUTLINE_DATA*/
         PLAN_ID: 52949
LAST_ACTIVE_TIME: 2024-03-19 17:14:44.234663
       QUERY_SQL: select * from t1 where c1 = 1 and c2 =  1
```

Troubleshooting SQL Performance Issues

Typical Scenarios of SQL Performance Issues - Hard Parsing Issues

Plans are frequently eliminated because the plan cache space is set too small.

You can increase the system variable ob_plan_cache_percentage to set the percentage of tenant memory occupied by the plan cache.

```
show variables like 'ob_plan_cache_percentage';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| ob_plan_cache_percentage | 5     |
+--------------------------+-------+
1 row in set (0.010 sec)
```

- Modify tenant system variables ob_plan_cache_percentage

```
set global ob_plan_cache_percentage = 10;

show variables like 'ob_plan_cache_percentage';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| ob_plan_cache_percentage | 10    |
+--------------------------+-------+
```

The maximum memory the plan cache can use = tenant memory limit * ob_plan_cache_percentage / 100. The default value is 5.

OCEANBASE

FROM INTRODUCTION TO PRACTICE

Thank You!

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

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