Alibaba Cloud MaxCompute

Best Practices

Issue: 20180803

Legal disclaimer

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

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

secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).

6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

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

Contents

1 SQL implements multiple rows of data into one

This article will show you how to use SQL to compress multiple data into one.

Scenario example

The following table data is used as an example:

class	gender	name
1	M	LiLei
1	F	HanMM
1	M	Jim
2	F	Kate
2	M	Peter

Scenario 1

Depending on your needs, the common scenarios are as follows:

class	names
1	LiLei,HanMM,Jim
2	Kate,Peter

Using a separator for string mosaic is similar to this, you can use the following statement:

```
SELECT class, wm_concat(',', name) FROM students GROUP BY class;
```

Scenario two

Another common requirement is as follows:

class	cnt_m	cnt_f
1	2	1
2	1	1

You can use the following statement for the need to change multiple columns like this:

```
SELECT
class
,SUM(CASE WHEN gender = 'M' THEN 1 ELSE 0 END) AS cnt_m
,SUM(CASE WHEN gender = 'F' THEN 1 ELSE 0 END) AS cnt_f
FROM students
```

```
GROUP BY class;
```

2 Group out the first n sections of each group of data

This article will show you how to group data, take out the first n sections of data for each data group.

Sample Data

Current data, as shown in the following table:

empno	ename	job	sal
7369	SMITH	CLERK	800.0
7876	SMITH	CLERK	1100.0
7900	JAMES	CLERK	950.0
7934	MILLER	CLERK	1300.0
7499	ALLEN	SALESMAN	1600.0
7654	MARTIN	SALESMAN	1250.0
7844	TURNER	SALESMAN	1500.0
7521	WARD	SALESMAN	1250.0

Implementation Method

You can achieve this in two ways:

- Take out the line number of each data and filter it with the where statement.

```
SELECT * FROM (
  SELECT empno
  , ename
  ,sal
  , job
  , ROW_NUMBER() OVER (PARTITION BY job ORDER BY sal) AS rn
  FROM emp
) tmp
WHERE rn < 10;
```

- Implement the Split Function using UDTF.

  For more information, see the last example in *this article*. This example can more quickly judge the current sequence number. In this example, it will not work if you have exceeded the planned number of lines (for example, 10), and thus improve the efficiency of the calculation.

3 Export SQL operation result

This article provides examples to illustrate how to download the MaxCompute SQL computing results by using several methods.

Note:

The Java SDK is used as an example throughout this article.

You can use one of the following methods to export the SQL statement execution results:

- If the data volume is small, use *SQL Task* to list all query results.
- If you want to export the results of a specific table or partition, use *Tunnel*.
- If the SQL statements are complex, use Tunnel and SQL Task in combination.
- *DataWorks* allows you to conveniently run SQL statements and *Synchronize data*. It supports regular scheduling and task dependency configuration.
- The open source tool **DataX** allows you to export data from your MaxCompute instance to the target data source with ease. For more information, see *DataX overview*.

Use SQL Task to export data

SQL Task is an interface of MaxCompute that the SDK can call directly to run SQL statements and return results conveniently.

`SQLTask.getResult(i);` returns a list which can be iterated cyclically to obtain the complete SQL computing results. This method has a defect. For more information, see *Other Operations* for the description of `SetProject READ_TABLE_MAX_ROW`.

Currently, you can adjust the maximum number of data records that the SELECT statement returns to the client up to **10,000**. If you run the SELECT statement on a client or use SQL Task, the query results are appended with Limit N. Limit N does not apply to the CREATE TABLE XX AS SELECT statement or in the case that the results are solidified in a specific table using INSERT INTO/OVERWRITE TABLE.

Use Tunnel to export data

If the query results to be exported are the full content of a table or a partition, you can use Tunnel to export the results. For more information, see *Command line tool* and *Tunnel SDK* which is compiled based on SDK.

An example is provided to illustrate how to export data by using the Tunnel command line. You can compile the Tunnel SDK only when data cannot be exported using some command lines. For more information, see *Batch data tunnel overview*.

```
tunnel d wc_out c:\wc_out.dat;
2016-12-16 19:32:08 - new session: 201612161932082d3c9b0a012f68e7
total lines: 3
2016-12-16 19:32:08 - file [0]: [0, 3), c:\wc_out.dat
downloading 3 records into 1 file
2016-12-16 19:32:08 - file [0] start
2016-12-16 19:32:08 - file [0] OK. total: 21 bytes
download OK
```

Use SQL Task and Tunnel to export data

SQL Task cannot export more than 10,000 records, whereas Tunnel can. You can use them in combination to export data.

The sample code is as follows:

```
private static final String accessId = "userAccessId";
private static final String accessKey = "userAccessKey";
private static final String endPoint = "http://service.odps.aliyun.com
/api";
private static final String project = "userProject";
private static final String sql = "userSQL";
private static final String table = "Tmp_" + UUID.randomUUID().
toString().replace("-", "_");//The name of the temporary table is a
random string.
private static final Odps odps = getOdps();
public static void main(String[] args) {
System.out.println(table);
runSql();
tunnel();
}
/*
* Download the results returned by SQL Task.
* */
private static void tunnel() {
TableTunnel tunnel = new TableTunnel(odps);
try {
DownloadSession downloadSession = tunnel.createDownloadSession(
project, table);
System.out.println("Session Status is : "
+ downloadSession.getStatus().toString());
long count = downloadSession.getRecordCount();
System.out.println("RecordCount is: " + count);
RecordReader recordReader = downloadSession.openRecordReader(0,
count);
Record record;
while ((record = recordReader.read()) ! = null) {
consumeRecord(record, downloadSession.getSchema());
}
recordReader.close();
} catch (TunnelException e) {
```

```
e.printStackTrace();
} catch (IOException e1) {
e1.printStackTrace();
}
}
/*
* Save the data record.
* If the data volume is small, you can print and copy the data
directly.  In reality, you can use Java.io to write the data to a
local file or a remote data storage.
* */
private static void consumeRecord(Record record, TableSchema schema) {
System.out.println(record.getString("username")+","+record.getBigint("
cnt"));
}
/*
* Run an SQL statement to save the query results to a temporary table
. The saved results can be downloaded using Tunnel.
* The lifecycle of the saved data is 1 day. The data does not occupy
much storage space even when an error occurs while you delete the data
.
* */
private static void runSql() {
Instance i;
StringBuilder sb = new StringBuilder("Create Table ").append(table)
.append(" lifecycle 1 as ").append(sql);
try {
System.out.println(sb.toString());
i = SQLTask.run(getOdps(), sb.toString());
i.waitForSuccess();
} catch (OdpsException e) {
e.printStackTrace();
}
}
/*
* Initialize the connection information of the MaxCompute (formerly
ODPS) instance.
* */
private static Odps getOdps() {
Account account = new AliyunAccount(accessId, accessKey);
Odps odps = new Odps(account);
odps.setEndpoint(endPoint);
odps.setDefaultProject(project);
return odps;
}
```

Use DataWorks to export data using synchronization

Using the preceding method, you can save the downloaded data. Other methods are required to create the data and implement the scheduling dependency between data creation and storage.

DataWorks allows you to *Configure a data synchronization task* and configure *Periodic running* and *Dependency among multiple tasks* to complete the process from data creation to storage.

An example is provided to illustrate how to use Data IDE to run SQL statements and configure a data synchronization task to create and export data.

Procedure

1. Create a workflow with an SQL node and a data synchronization node. Connect the two nodes and configure an inter-node dependency, with the SQL node as the data production node and the data synchronization node as the data export node.
2. Configure the SQL node.

   Note:
   Run an SQL statement to create a table before you configure synchronization. If no table exists, the synchronization task cannot be configured.
3. Perform the following to configure the data synchronization task.

   a. Select a Source.

   b. Select a Target.

   c. Map fields.

   d. Control the tunnel.

   e. Preview and Save.
4. After workflow scheduling is configured, save and submit the workflow. Click **Test Run**. If you do not configure workflow scheduling, you can use the default scheduling configuration directly. View the running log on data synchronization as as in the following figure.

```
2016-12-17 23:43:46.394 [job-15598025] INFO JobContainer -
Task start time : 2016-12-17 23:43:34
Task end time : 2016-12-17 23:43:46
Total task time : 11s
Average data per task : 31.36 KB/s
Write speed : 1,668 rec/s
Read records : 16,689
Failed read-write attempts : 0
```

5. Run an SQL statement to view the data synchronization results.

4 Demo: Modify incompatible SQL

The MaxCompute development team has completed the grayscale upgrade of MaxCompute 2.0 recently. The new version fully embraces open source ecosystems, supports more languages and functions, and enables faster operation. It also implements more rigorous syntax inspection, so that errors may be returned for some less rigorous syntax cases that can run normally in the earlier editor.

To enable smooth grayscale upgrade of MaxCompute 2.0, the MaxCompute framework supports rollback. If a task of MaxCompute 2.0 fails, it is executed in MaxCompute 1.0. The rollback increases the E2E latency of the task. We recommend that you set `set odps.sql.planner.mode=lot;` to manually disable the rollback function before submitting jobs, to avoid the impact resulting from the changes made to the MaxCompute rollback policy.

The MaxCompute team notifies the owners of problematic tasks by email or DingTalk based on the online rollback condition. Modify your SQL tasks immediately; otherwise, the tasks may fail. Check the following errors against your tasks to avoid task failure in case of missed notifications.

The following lists the syntaxes for which MaxCompute 2.0 may return errors.

group.by.with.star

SELECT * … GROUP BY… syntax is problematic.

In the earlier version of MaxCompute, `select * from group by key` is supported even when the columns that match * are not included in the GROUP BY key. Compatible with Hive, MaxCompute 2.0 prohibits this syntax unless the GROUP BY list is a column in all source tables. For example:

Scenario 1: **The GROUP BY key does not include all columns**.

Incorrect syntax:

```
SELECT * FROM t GROUP BY key;
```

Error message:

```
FAILED: ODPS-0130071:[1,8] Semantic analysis exception - column
reference t.value should appear in GROUP BY key
```

Correct syntax:

```
SELECT DISTINCT key FROM t;
```

Scenario 2: **The GROUP BY key includes all columns**.

Syntax that is not recommended:

```
SELECT * FROM t GROUP BY key, value; -- t has columns key and value
```

Though MaxCompute 2.0 does not return an error, we recommend that you modify the syntax as follows:

```
SELECT DISTINCT key, value FROM t;
```

bad.escape

The escape sequence is incorrect.

According to the MaxCompute documentation, in string literal, every ASCII character ranging from 0 to 127 must be written in the format of a backslash followed by three octal numbers. For example, 0 is written as \001, and 1 is written as \002. Currently, \01 and \0001 are considered as \001.

It may bring confusions to new users. For example, “\000” + “1” cannot written as “\0001”. For users who migrate data from other systems to MaxCompute, incorrect data may be generated.

Note:

An error may be returned if numbers are appended to `\0 00`, for example, `\0001 - \0009 or \00001`.

MaxCompute 2.0 solves this problem. The script owner must correct the sequence, for example:

Incorrect syntax:

```
SELECT split(key, "\01"), value like "\0001" FROM t;
```

Error message:

```
FAILED: ODPS-0130161:[1,19] Parse exception - unexpected escape
sequence: 01
ODPS-0130161:[1,38] Parse exception - unexpected escape sequence: 0001
```

Correct syntax:

```
SELECT split(key, "\001"), value like "\001" FROM t;
```

column.repeated.in.creation

Duplicate column names are detected during execution of the CREATE TABLE statement.

MaxCompute 2.0 returns an error when duplicate column names are detected when the CREATE TABLE statement is executed. For example:

Incorrect syntax:

```
CREATE TABLE t (a BIGINT, b BIGINT, a BIGINT);
```

Error message:

```
FAILED: ODPS-0130071:[1,37] Semantic analysis exception - column
repeated in creation: a
```

Correct syntax:

```
CREATE TABLE t (a BIGINT, b BIGINT);
```

string.join.double

The data on both sides of the equal sign of a Join condition belongs to the String and Double types.

In the old version of MaxCompute, the String and Double type data is converted to the Bigint type at the cost of precision. 1.1 = “1” in a Join condition is considered as equal. Compatible with Hive , MaxCompute 2.0 converts the String and Double type data to the Double type.

Syntax that is not recommended:

```
SELECT * FROM t1 JOIN t2 ON t1.double_value = t2.string_value;
```

Warning:

```
WARNING:[1,48]  implicit conversion from STRING to DOUBLE, potential
data loss, use CAST function to suppress
```

Recommended correction:

```
select * from t1 join t2 on t.double_value = cast(t2.string_value as
double);
```

You can also convert the data as needed.

window.ref.prev.window.alias

Window functions reference other aliases in the select list of the same level.

For example:

With rn absent from t1, the incorrect syntax is as follows:

```
SELECT row_number() OVER (PARTITION BY c1 ORDER BY c1) rn,
row_number() OVER (PARTITION by c1 ORDER BY rn) rn2
FROM t1;
```

Error message:

```
FAILED: ODPS-0130071:[2,45] Semantic analysis exception - column rn
cannot be resolved
```

Correct syntax:

```
SELECT row_number() OVER (PARTITION BY c1 ORDER BY rn) rn2
FROM

SELECT c1, row_number() OVER (PARTITION BY c1 ORDER BY c1) rn
FROM t1
) tmp;
```

select.invalid.token.after.star

Select * is followed by an alias.

The select list allows the use of * to select all the columns of a table, but * cannot be followed by any alias even if the expanded * has only one column. The new editor returns errors for similar syntaxes. For example:

Incorrect syntax:

```
select * as alias from dual;
```

Error message:

```
FAILED: ODPS-0130161:[1,10] Parse exception - invalid token 'as'
```

Correct syntax:

```
select * from dual;
```

agg.having.ref.prev.agg.alias

The select list is preceded by an aggregate function alias when HAVING exists. For example:

Incorrect syntax:

```
SELECT count(c1) cnt,
sum(c1) / cnt avg
FROM t1
GROUP BY c2
HAVING cnt > 1;
```

Error message:

```
FAILED: ODPS-0130071:[2,11] Semantic analysis exception - column cnt
cannot be resolved
ODPS-0130071:[2,11] Semantic analysis exception - column reference cnt
 should appear in GROUP BY key
```

s and cnt do not exist in source table t1, but MaxCompute of the old version does not return an error because HAVING exists. In MaxCompute 2.0, the prompt “column cannot be resolved” appears and an error is returned.

Correct syntax:

```
SELECT cnt, s, s/cnt avg
FROM

SELECT count(c1) cnt,
sum(c1) s
FROM t1
GROUP BY c2
HAVING count(c1) > 1
```

```
) tmp;
```

order.by.no.limit

ORDER BY is not followed by the LIMIT statement.

By default, MaxCompute requires that ORDER BY be followed by the LIMIT statement to limit the number of data records. Because ORDER BY is used for full data sorting, the execution performance is low without the LIMIT statement. For example:

Incorrect syntax:

```
select * from (select *
from (select cast(login_user_cnt as int) as uv, '3' as shuzi
from test_login_cnt where type = 'device' and type_name = 'mobile') v
order by v.uv desc) v
order by v.shuzi limit 20;
```

Error message:

```
FAILED: ODPS-0130071:[4,1] Semantic analysis exception - ORDER BY must
 be used with a LIMIT clause
```

Correct syntax:

Add the LIMIT statement to the subquery “order by v.uv desc”.

In MaxCompute 1.0, the view inspection is not strict. For example, a view is created in a project which does not require LIMIT statement check (odps.sql.validate.orderby.limit=false).

```
CREATE VIEW dual_view AS SELECT id FROM dual ORDER BY id;
```

When you run the following statement to access the view:

```
SELECT * FROM dual_view;
```

MaxCompute 1.0 does not return an error, whereas MaxCompute 2.0 returns the following error:

```
FAILED: ODPS-0130071:[1,15] Semantic analysis exception - while
resolving view xdj.xdj_view_limit - ORDER BY must be used with a LIMIT
 clause
```

generated.column.name.multi.window

Automatically generated aliases are used.

In the earlier version of MaxCompute, an alias is auto generated for every expression of the SELECT statement. The alias is displayed on the console. However, the earlier version does not

guarantee that the alias generation rule is correct or remains unchanged. Therefore, we do not recommend that you use auto generated aliases.

In MaxCompute 2.0, a warning is given for use of auto generated aliases, but due to breadth of involvement such use is not prohibited for the moment.

In some cases, known changes are made to the alias generation rules in the different versions of MaxCompute. Some online jobs depend on the auto generated aliases. Queries may fail when MaxCompute performs version upgrade or rollback. If you have such problems, modify queries and explicitly specify the alias of the column of interest. For example:

Not recommended syntax:

```
SELECT _c0 FROM (SELECT count(*) FROM dual) t;
```

Recommended correction:

```
SELECT c FROM (SELECT count(*) c FROM dual) t;
```

non.boolean.filter

Non-Boolean filter conditions are used.

MaxCompute prohibits the implicit conversion between the Boolean type and other data types. However, the earlier version of MaxCompute allows the use of Bigint type filter conditions in some cases. MaxCompute 2.0 prohibits the use of Bigint type filter conditions. If your scripts have Bigint type filter conditions, modify them promptly. For example:

Incorrect syntax:

```
select id, count(*) from dual group by id having id;
```

Error message:

```
FAILED: ODPS-0130071:[1,50] Semantic analysis exception - expect a
BOOLEAN expression
```

Correct syntax:

```
select id, count(*) from dual group by id having id <> 0;
```

post.select.ambiguous

The GROUP BY, CLUSTER BY, DISTRIBUTE BY, and SORT BY statements reference columns with conflicting names.

In the old version of MaxCompute, by default, the system selects the last column of the select list as the operation object. MaxCompute 2.0 reports an error in this case. Make relevant modification timely. For example:

Incorrect syntax:

```
select a, b as a from t order by a limit 10;
```

Error message:

```
FAILED: ODPS-0130071:[1,34] Semantic analysis exception - a is
ambiguous, can be both t.a or null.a
```

Correct syntax:

```
select a as c, b as a from t order by a limit 10;
```

The pushed change covers the statements with conflicting column names but the same syntax. Although there is no ambiguity, an error is often returned for these statements and a warning is triggered. We recommend that you make relevant modification.

duplicated.partition.column

Partitions with the same name are specified in a query.

In the earlier version of MaxCompute, no error is returned when two partition keys with the same name are specified. The latter partition key overwrites the former one. It causes confusion. MaxCompute 2.0 returns an error in this case.

Incorrect syntax:

```
insert overwrite table partition (ds = '1', ds = '2') select ...
```

In fact, ds = ‘1’ is ignored during execution.

Correct syntax:

```
insert overwrite table partition (ds = '2')select ...
```

Incorrect syntax:

```
create table t (a bigint, ds string) partitioned by (ds string);
```

Correct syntax:

```
create table t (a bigint) partitioned by (ds string);
```

order.by.col.ambiguous

The ORDER BY clause references the duplicate aliases in the select list.

Incorrect syntax:

```
SELECT id, id
FROM dual
ORDER BY id;
```

Correct syntax:

```
SELECT id, id id2
FROM dual
ORDER BY id;
```

Remove the duplicate aliases before the ORDER BY clause can reference them.

in.subquery.without.result

colx does not exist in the source table if colx in subquery does not return any results.

Incorrect syntax:

```
SELECT * FROM dual
WHERE not_exist_col IN (SELECT id FROM dual LIMIT 0);
```

Error message:

```
FAILED: ODPS-0130071:[2,7] Semantic analysis exception - column
not_exist_col cannot be resolved
```

ctas.if.not.exists

The syntax of the target table is incorrect.

If the target table exists, the earlier version of MaxCompute does not check the syntax, whereas MaxCompute 2.0 does. Many errors may return in this case, for example:

Incorrect syntax:

```
CREATE TABLE IF NOT EXISTS dual
AS
SELECT * FROM not_exist_table;
```

Error message:

```
FAILED: ODPS-0130131:[1,50] Table not found - table meta_dev.
not_exist_table cannot be resolved
```

worker.restart.instance.timeout

In the earlier version of MaxCompute, every time a UDF outputs a record, a write operation is triggered on the distributed file system, and a heartbeat packet is sent to Fuxi. If the UDF does not output any records for 10 minutes, the following error is reported:

```
FAILED: ODPS-0123144: Fuxi job failed - WorkerRestart errCode:252,
errMsg:kInstanceMonitorTimeout, usually caused by bad udf performance.
```

The runtime framework of MaxCompute 2.0 supports vectoring. It processes multiple rows of a column at a time to improve the execution efficiency. Vectoring may cause the normal statements to time out in the case that a heartbeat packet is not sent to Fuxi within the specified time while multiple records are processed at a time. The interval between two output records does not exceed 10 minutes.

If a time-out error occurs, we recommend that you first check the UDF performance. It takes several seconds to process each record. If the UDF performance cannot be optimized, as a workaround, you can set the value of “batch row” manually. The default value is 1024.

```
set odps.sql.executionengine.batch.rowcount=16;
```

divide.nan.or.overflow

The old version of MaxCompute does not support division constant folding.

In the old version of MaxCompute, the physical execution plan for a statement is as follows:

```
EXPLAIN
Select if (false, 0/0, 1.0)
FROM dual;
In Task M1_Stg1:
    Data source: meta_dev.dual
```

```
    TS: alias: dual
      SEL: If(False, Divide(UDFToDouble(0), UDFToDouble(0)), 1.0)
        FS: output: None
```

The IF and Divide functions are retained. During execution, the first parameter of IF is set to “false ”, and the expression of the second parameter Divide is not evaluated. Division by zero is normal.

MaxCompute 2.0 supports division constant folding. An error is returned. For example:

Incorrect syntax:

```
SELECT IF(FALSE, 0/0, 1.0)
FROM dual;
```

Error message:

```
FAILED: ODPS-0130071:[1,19] Semantic analysis exception - encounter
 runtime exception while evaluating function /, detailed message:
DIVIDE func result NaN, two params are 0.000000 and 0.000000
```

An overflow error may occur besides nan. For example:

Incorrect syntax:

```
SELECT IF(FALSE, 1/0, 1.0)
FROM dual;
```

Error message:

```
FAILED: ODPS-0130071:[1,19] Semantic analysis exception - encounter
 runtime exception while evaluating function /, detailed message:
DIVIDE func result overflow, two params are 1.000000 and 0.000000
```

Correct syntax:

We recommend that you remove /0 and use valid constants.

A similar problem occurs in the constant folding of CASE WHEN, such as CASE WHEN TRUE THEN 0 ELSE 0/0. During constant folding in MaxCompute 2.0, all subexpressions are evaluated , causing incorrect division by zero.

CASE WHEN may involve more complex optimization scenarios, for example:

```
SELECT CASE WHEN key = 0 THEN 0 ELSE 1/key END
FROM (
SELECT 0 AS key FROM src
UNION ALL
```

```
SELECT key FROM src) r;
```

The optimizer pushes down the division operation to subqueries. A similar conversion is as follows :

```
M (
SELECT CASE WHEN 0 = 0 THEN 0 ELSE 1/0 END c1 FROM src
UNION ALL
SELECT CASE WHEN key = 0 THEN 0 ELSE 1/key END c1 FROM src) r;
```

Error message:

```
FAILED: ODPS-0130071:[0,0] Semantic analysis exception - physical plan
 generation failed: java.lang.ArithmeticException: DIVIDE func result
overflow, two params are 1.000000 and 0.000000
```

An error is returned for the constant folding of the first clause of UNION ALL. We recommend that you transfer CASE WHEN in SQL to subqueries and eliminate useless CASE WHEN statements and /0.

```
SELECT c1 END
FROM (
SELECT 0 c1 END FROM src
UNION ALL
SELECT CASE WHEN key = 0 THEN 0 ELSE 1/key END) r;
```

small.table.exceeds.mem.limit

The earlier version of MaxCompute supports Multi-way Join optimization. Multiple JOIN statements with the same Join key are merged for execution in the same Fuxi task, such as J4_1_2_3_Stg1 in the following example.

```
EXPLAIN
SELECT t1.
FROM t1 JOIN t2 ON t1.c1 = t2.c1
JOIN t3 ON t1.c1 = t3.c1;
```

The earlier version of MaxCompute has the following physical execution plan:

```
In Job job0:
root Tasks: M1_Stg1, M2_Stg1, M3_Stg1
J4_1_2_3_Stg1 depends on: M1_Stg1, M2_Stg1, M3_Stg1

In Task M1_Stg1:
    Data source: meta_dev.t1

In Task M2_Stg1:
    Data source: meta_dev.t2
```

```
In Task M3_Stg1:
    Data source: meta_dev.t3

In Task J4_1_2_3_Stg1:
    JOIN: t1 INNER JOIN unknown INNER JOIN unknown
        SEL: t1._col0, t1._col1, t1._col2
            FS: output: None
```

The earlier version of MaxCompute still keeps the physical execution plan when MapJoin Hints are added, and gives priority to applications in Multi-way Join optimization. It may ignore the user-specified MapJoin Hint.

```
EXPLAIN
SELECT /*+mapjoin(t1)*/ t1.
FROM t1 JOIN t2 ON t1.c1 = t2.c1
JOIN t3 ON t1.c1 = t3.c1;
```

The physical execution plan of the earlier version of MaxCompute is the same as the preceding one.

The optimizer of MaxCompute 2.0 gives priority to the user-specified MapJoin Hint. In the preceding example, if the value of t1 is relatively higher, the following error is returned:

```
FAILED: ODPS-0010000:System internal error - SQL Runtime Internal
 Error: Hash Join Cursor HashJoin_REL… small table exceeds, memory
 limit(MB) 640, fixed memory used …, variable memory used …
```

We recommend that you remove the MapJoin Hint if MapJoin is not your expected behavior.

sigkill.oom

Like small.table.exceeds.mem.limit, if you specify a MapJoin Hint and your small tables are of a relatively larger size, in the earlier version of MaxCompute, multiple JOIN statements may be optimized by Multi-way Join and can be successfully executed. However, in MaxCompute 2.0, some users may set odps.sql.mapjoin.memory.max to prevent small tables from exceeding the size limit. Each MaxCompute worker has a fixed memory limit. If small tables are of a large size, MaxCompute workers may be killed because the memory limit exceeds. The error is similar to the following:

```
Fuxi job failed - WorkerRestart errCode:9,errMsg:SigKill(OOM), usually
 caused by OOM(outof memory).
```

We recommend that you remove MapJoin Hint and use Multi-way Join.

wm_concat.first.argument.const

According to the *Aggregate function* document, the first parameter of WM_CONCAT must be a constant. The old version of MaxCompute does not have strict check standards. For example, when the source table has no data, no error is returned even if the first parameter of WM_CONCAT is ColumnReference.

```
Function declaration:
string wm_concat(string separator, string str)
Parameter description:
Separator: String-type constant.  Constants of other types or non-
constants can cause exceptions.
```

MaxCompute 2.0 checks the validity of parameters during the planning phase. An error is returned if the first parameter of WM_CONCAT is not a constant. For example:

Incorrect syntax:

```
SELECT wm_concat(value, ',') FROM src GROUP BY value;
```

Error message:

```
FAILED: ODPS-0130071:[0,0] Semantic analysis
 exception - physical plan generation failed:
 com.aliyun.odps.lot.cbo.validator.AggregateCallValidator
$AggregateCallValidationException: Invalid argument type - The first
 argument of WM_CONCAT must be constant string.
```

pt.implicit.convertion.failed

srcpt is a partition table with two partitions:

```
CREATE TABLE srcpt(key STRING, value STRING) PARTITIONED BY (pt STRING
);
ALTER TABLE srcpt ADD PARTITION (pt='pt1');
ALTER TABLE srcpt ADD PARTITION (pt='pt2');
```

For the preceding SQL statements, the constants of the IN INT type in the pt columns of the String type are converted to the Double type for comparison. Even if odps.sql.udf.strict.mode is set to “true” in the project, the old version of MaxCompute does not return an error and it filters out all pt columns, whereas in MaxCompute 2.0, an error is returned. For example:

Incorrect syntax:

```
SELECT key FROM srcpt WHERE pt IN (1, 2);
```

Error message:

```
FAILED: ODPS-0130071:[0,0] Semantic analysis exception - physical
plan generation failed: java.lang.NumberFormatException: ODPS-0123091
:Illegal type cast - In function cast, value 'pt1' cannot be casted
from String to Double.
```

We recommend that you avoid comparing String type partition columns and INT-type constants and convert INT-type constants to the String type.

having.use.select.alias

SQL specifies that the GROUP BY+HAVING clause precedes the SELECT clause. Therefore, the column alias generated by the SELECT clause cannot be used in the HAVING clause. For example:

Incorrect syntax:

```
SELECT id id2 FROM DUAL GROUP BY id HAVING id2 > 0;
```

Error message:

```
FAILED: ODPS-0130071:[1,44] Semantic analysis exception - column id2
 cannot be resolvedODPS-0130071:[1,44] Semantic analysis exception -
column reference id2 should appear in GROUP BY key
```

id2 is the column alias generated by the SELECT clause and cannot be used in the HAVING clause.

dynamic.pt.to.static

In MaxCompute 2.0, dynamic partitions may be converted to static ones by the optimizer. For example:

```
INSERT OVERWRITE TABLE srcpt PARTITION(pt) SELECT id, 'pt1' FROM dual;
```

is converted to:

```
INSERT OVERWRITE TABLE srcpt PARTITION(pt='pt1') SELECT id FROM dual;
```

If the specified partition value is invalid (for example, '${bizdate}' is used), MaxCompute 2.0 returns an error during syntax check. For more information, see *MaxCompute partition value definition*.

Incorrect syntax:

```
INSERT OVERWRITE TABLE srcpt PARTITION(pt) SELECT id, '${bizdate}'
FROM dual LIMIT 0;
```

Error message:

```
FAILED: ODPS-0130071:[1,24] Semantic analysis exception - wrong
columns count 2 in data source, requires 3 columns (includes dynamic
partitions if any)
```

In the earlier version of MaxCompute, with LIMIT 0, no results are returned by the SQL statements , and no dynamic partitions are created. In this case, no error is returned.

lot.not.in.subquery

Processing of the null value of In subquery.

In the standard SQL IN operation, if the value list contains a null value, the returned value may be “null” or “true”, but cannot be “false”. For example, 1 in (null, 1, 2, 3) is “true”, whereas 1 in (null, 2, 3) is “null”, and null in (null, 1, 2, 3) is “null”. Likewise, for the NOT IN operation, if the value list contains a null value, the returned value may be “false” or “null”, but cannot be “true”.

MaxCompute 2.0 performs processing with a standard behavior. If you receive a notification on this problem, check your queries to determine whether the subqueries in the IN operation have a null value and whether the related behavior meets your expectation. If not, make relevant changes . For example:

```
select * from t where c not in (select accepted from c_list);
```

Ignore this problem if “accepted” does not have null values. If a null value exists, c not in (select accepted from c_list) returns the value “true” in the earlier version, but returns a null value in the new version.

Correct syntax:

```
select * from t where c not in (select accepted from c_list where
accepted is not null)
```

6 Optimize long tail computing

Long tail is a common and difficult problem in distributed computing. The workload of each node is different when data is distributed unevenly. The whole task does not end until the work on the slowest node is complete.

To address this problem, you can distribute a piece of task to multiple workers for execution, rather than designate a worker to run the heaviest task. This article describes how to address typical cases of long tail.

JOIN long tail

Cause

When the key of a JOIN statement has a large amount of data, a long tail occurs.

Solution

You can solve the problem in three steps:

- Verify that one or both of the tables are not small tables. If one table is large and the other small, you can use the MAPJOIN statement to cache the small table. For the syntax and relevant description, see *SELECT operation*. If the job is a MapReduce job, you can use the resource table function to cache the small table.
- If both tables are relatively large, reduce duplicated data as much as possible.
- If the problem persists, consider service optimization to avoid the calculation of Cartesian product on the two keys with a large data volume.

Group By long tail

Cause

When a key of the Group By statement has a large amount of computations, a long tail occurs.

Solution

You can solve the problem in either of the following ways:

- Rewrite the SQL statement and add random numbers to split the long key. For example:

```
Select   Key,Count(*)  As  Cnt  From  TableName  Group  By  Key;
```

Provided that the Combiner is skipped, data is shuffled from the M node to the R node. Then , the R node performs the COUNT operation. The execution plan is M > R. If you want to redistribute work to the key with the long tail, modify the statement as follows:

```
-- Assume that the key with the long tail is KEY001.
SELECT a.Key
  , SUM(a.Cnt) AS Cnt
FROM (
  SELECT Key
    , COUNT(*) AS Cnt
FROM TableName
GROUP BY Key,
  CASE
    WHEN Key = 'KEY001' THEN Hash(Random()) % 50
    ELSE 0
   END
) a
GROUP BY a.Key;
```

The execution plan is changed to M > R > R. The execution lengthens, but the consumed time may reduce because the long-tail key is processed in two steps.

Note:

If you use the preceding method to add an R execution step for solving a long tail problem that is not serious, the consumed time may increase.

- Set system parameters as follows:

```
set odps.sql.groupby.skewindata=true。
```

Parameter setting is a universal optimization method, but the results are not satisfying because this method does not consider specific services. You can rewrite the SQL statement in a more efficient manner based on the actual data.

Distinct long tail

When a long tail occurs in a Distinct statement, the key splitting method does not apply. You can consider other methods.

Solution

```
--Original SQL statement, with the null UID skipped
SELECT COUNT(uid) AS Pv
```

```
    , COUNT(DISTINCT uid) AS Uv
FROM UserLog;
```

Rewrite the statement as follows:

```
SELECT SUM(PV) AS Pv
    , COUNT(*) AS UV
FROM (
    SELECT COUNT(*) AS Pv
      , uid
    FROM UserLog
    GROUP BY uid
) a;
```

The Distinct statement is rewritten to a Count statement to relieve the computing pressure on a single Reducer. This solution also enables Group By statement optimization and Combiner execution, greatly improving the performance.

Long tail of a dynamic partition

Cause

- To sort the data of small files, the dynamic partition function starts a Reduce task in the final stage of execution. A long tail occurs when the data written by the dynamic partition function is skewed.
- Misuse of the dynamic partition function often results in long tails.

Solution

If the target partition for data write is determined, you can specify this partition during the Insert operation, instead of using the dynamic partition function.

Remove long tails using the Combiner

For MapReduce jobs, it is a common practice to use the Combiner to remove long tails. This practice has been mentioned in the WordCount example. The Combiner can reduce the volume of data shuffle from the Mapper to the Reducer, thus sufficiently reduce the overhead of network transfer. The optimization is automatically done for MaxCompute SQL statements.

 **Note:**

The Combiner only supports optimization for Map. You must make sure that the results of Combiner execution are the same. For example, in WordCount, the results are the same for two (KEY,1)s and one (KEY,2). For example, in average value calculation, (KEY,1) and (KEY,2) cannot be directly merged into (KEY,1.5) in the Combiner.

Remove long tails using system optimization

In addition to the Local Combiner, you can use the optimization method provided by MaxCompute to remove long tails. For example, the following content (+N backups) is logged during the task running process:

```
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047[100%]
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047[100%]
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047(+1 backups)[100%]
M1_Stg1_job0:0/521/521[100%] M2_Stg1_job0:0/1/1[100%] J9_1_2_Stg5_job0
:0/523/523[100%] J3_1_2_Stg1_job0:0/523/523[100%] R6_3_9_Stg2_job0:1/
1046/1047(+1 backups)[100%]
```

There total number of Reducers is 1,047, of which 1,046 are complete. The last one is incomplete. After the system identifies this condition, it starts a new Reducer to run the same data. Then, the system takes the data of the Reducer which completes first and combines the data to the result set.

Remove long tails using service optimization

The preceding optimization methods cannot solve all problems. You can analyze your services to find a better solution.

- A large amount of noisy data may exist in reality. For example, you want to check the behavior data in the access record of each user by visitor ID. You must remove crawler data first, even though crawlers become increasingly difficult to identify; otherwise, the crawler data may easily cause a long tail during the calculation process. A similar case is when you associate data by a specific ID, you must check whether the field for association is null.
- Special services exist. For example, ISV operation records differ greatly from common user records in terms of data volume and behavior. You can use a special method to analyze and handle the ISV operation records for major accounts independently.
- When data distribution is uneven, do not use a constant field as the Distribute by field for full sorting.

7 Reasonableness evaluation of partition pruning

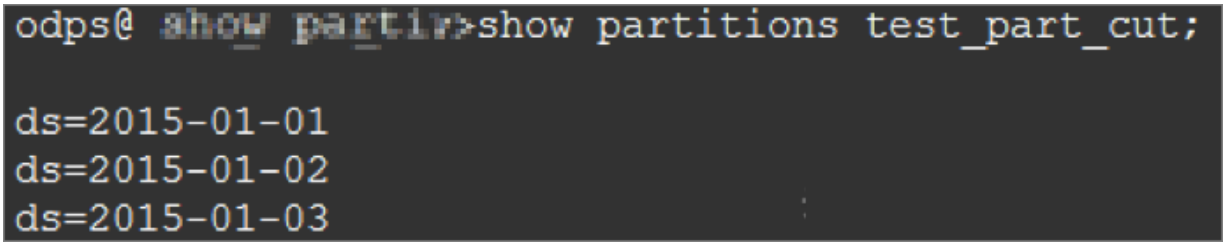
Background and purposes

Partition tables of MaxCompute refer to the partition spaces specified during table creation. This refers certain fields in the table as partition columns. When using data, if you mention the name of the partition you want to access, you can only read data from the corresponding partition. This eliminates need to scan the whole table, helping improve processing efficiency and reduce costs.

Partition pruning is to specify the filtering conditions for partition columns so that only part of partition data in the table is read during SQL execution. This prevents data errors and resource waste caused by full table scan. Partition pruning seems simple, but actually partition failure often occurs. This article uses examples to introduce how to solve common problems.

Problem examples

The following figure shows the partitions of the test table test_part_cut.

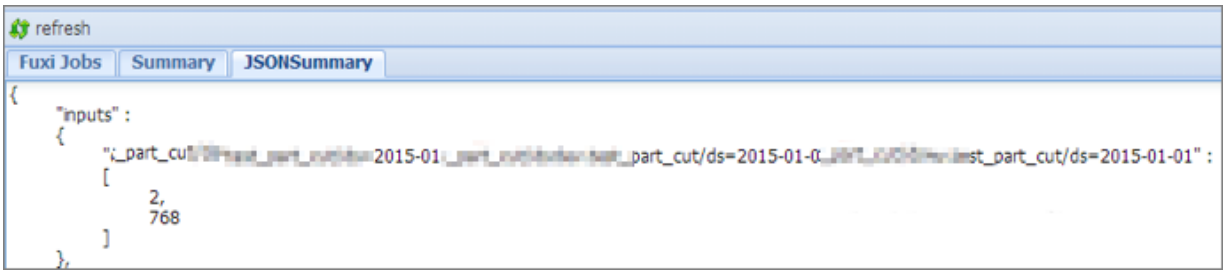


Run the following SQL code:

```
select count(*)
from test_part_cut
where ds= bi_week_dim('20150102');

--bi_week_dim is a user defined function. The return format is (year,
sequential number of the week).
--If the date is normal, the system checks whether the date belongs
to weeks of the input year when taking Thursday as the start day of a
 week. For example, if 20140101 is input, 2013,52 is returned because
 January 1, 2014 is Wednesday and considered as the last week of 2013.
  If 20150101 is input, 2015,1 is returned.
--If 20151231 is input, because December 31, 2015 is Thursday and is
in the same week as January 1, 2016, 2016,1 is returned.
```

The returned result of bi_week_dim('20150102') is 2015,1 and does not conform to the partition values of the test_part_cut table. Generally, we think that the preceding SQL statements do not read any partition, but, actually, **the SQL statements read all partitions in the test_part_cut table**. The following figure shows the LogView:



In the preceding figure, the SQL statements read all partitions in the test_part_cut table when executed.

The preceding example shows that though easy-to-use, partition pruning often fails. Therefore, this article focuses on the following aspects:

- Verify whether partition pruning in SQL statements takes effect.
- Know common scenarios that cause failed partition pruning.

Verify whether partition pruning takes effect

You can run the explain command and view SQL execution plans to check whether partition pruning in the SQL statements takes effect.

- Effect of failed partition pruning

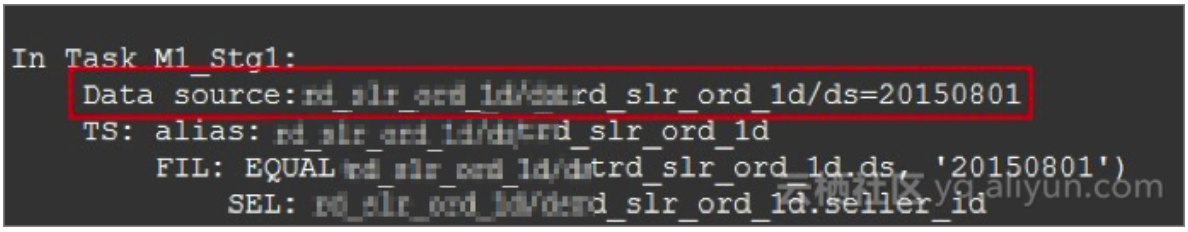
```
explain
select seller_id
from xxxxx_trd_slr_ord_1d
where ds=rand();
```

The red box in the preceding figure shows that all the 1,344 partitions of the xxxxx_trd_slr_ord_1d table are read.

- Effect of successful partition pruning

```
explain
select seller_id
from xxxxx_trd_slr_ord_1d
```

```
where ds='20150801';
```



The red box in the preceding figure shows that only the 20150801 partition of the xxxxx_trd_slr_ord_1d table is read.

Analysis on scenarios of failed partition pruning

Partition pruning sometimes fails when custom functions or part of system functions are used , and may also fail in the Where clause when the Join clause for joining is used. These two scenarios are explained in the following examples.

Failed partition pruning caused by custom functions

If partition pruning conditions contain custom functions, partition pruning fails. However, some system functions may also cause partition pruning to fail. Therefore, considering the limit on partition values, you must run the explain command to view the SQL statement execution plans and verify that partition pruning has taken effect when unusual functions are used.

```
explain
select ...
from xxxxx_base2_brd_ind_cw
where ds = concat(SPLIT_PART(bi_week_dim(' ${bdp.system.bizdate}'),
 ',', 1), SPLIT_PART(bi_week_dim(' ${bdp.system.bizdate}'), ',', 2))
```

In the preceding SQL statements, the custom functions used in partition pruning trigger full table scan.

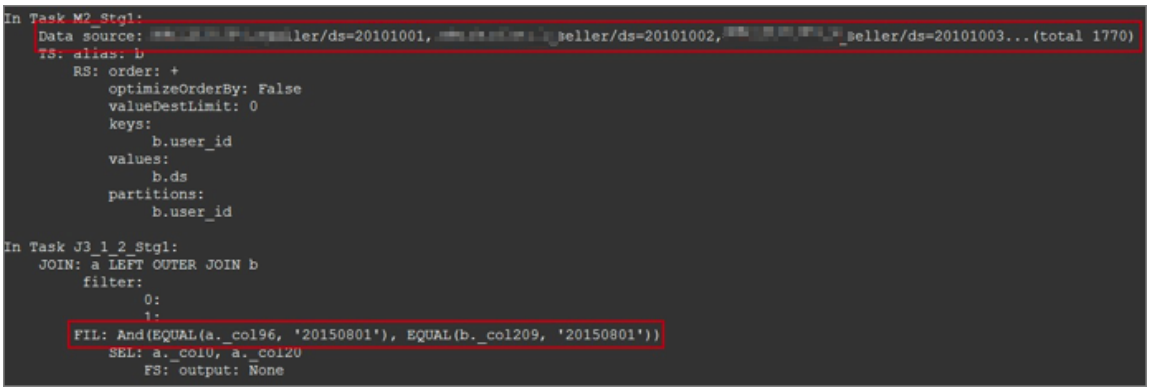
Failed partition pruning caused by the Join clause

In SQL statements, when you use the Join clause for joining, if partition pruning conditions are placed in the on clause, the partition pruning takes effect. If the conditions are placed in the Where clause, the partition pruning takes effect for the primary table, and does not apply to the foreign table. The three Join clauses are described as follows.

- **Left Outer Join**

- Place all partition pruning conditions in the on clause

```
explain
select a.seller_id
    ,a.pay_ord_pbt_1d_001
from xxxxx_trd_slr_ord_1d a
left outer join
     xxxxx_seller b
on a.seller_id=b.user_id
and a.ds='20150801'
and b.ds='20150801';
```



As shown in the preceding figure, the left table is under full table scan, and only partition pruning for the right table takes effect.

- Place partition pruning conditions in the where clause

```
explain
select a.seller_id
    ,a.pay_ord_pbt_1d_001
from xxxxx_trd_slr_ord_1d a
left outer join
    xxxxx_seller b
on a.seller_id=b.user_id
where a.ds='20150801'
```

```
and b.ds='20150801';
```

```
In Task M2_Stg1:
    Data source:seller/ds=seller/ds=20150801
    TS: alias: b
        FIL: EQUAL(b.ds, '20150801')
            RS: order: +
                optimizeOrderBy: False
                valueDestLimit: 0
                keys:
                     b.user_id
                values:
                partitions:
                     b.user_id

In Task J3_1_2_Stg1:
    JOIN: a LEFT OUTER JOIN unknown
         filter:
                0:
                1:
        SEL: a._col0, a._col20
            FS: output: None

In Task M1_Stg1:
    Data source:seller/ds=trd_slr_ord_1d/ds=20150801
    TS: alias: a
```

As shown in the preceding figure, partition pruning for both the two tables takes effect.

- **Right Outer Join**

  Similar to Left Outer Join, if partition pruning conditions are in the on clause, the conditions take effect only for the left table of Right Outer Join. If conditions are in the where clause, the conditions take effect for both the two tables.

- **Full Outer Join**

  Partition pruning conditions take effect only in the where clause, and fail in the on clause.

Impact and consideration

- Partition pruning failure has serious results, which can be hardly discovered by users. Therefore, we recommend that you check for failed partition pruning when submitting the code.
- More efforts are needed to solve the problem that custom functions cannot be used for partition pruning.

8 Learn to write SQL statements quickly

Using practical examples, this document introduces MaxCompute SQL and helps you understand how to write SQL statements and what is the difference between MaxCompute SQL and standard SQL. Read this article with *MaxCompute SQL basic documentation* for better understanding.

Prepare a dataset

In the example the Emp/Dept table is used as the dataset. For your convenience, we provide related MaxCompute table creation statements and data files (*emp table data files* and *dept table data files*). You can create tables on the MaxCompute projects and upload data.

The DDL statements for creating an Emp table are as follows:

```
CREATE TABLE IF NOT EXISTS emp (
  EMPNO string,
  ENAME string ,
  JOB string ,
  MGR bigint ,
  HIREDATE datetime ,
  SAL double ,
  COMM double ,
  DEPTNO bigint );
```

The DDL statements for creating a Dept table are as follows:

```
CREATE TABLE IF NOT EXISTS dept (
  DEPTNO bigint ,
  DNAME string,
  LOC string);
```

SQL operation

Notes for SQL beginners

- If you use Group by, the Select part must be either group items or aggregate functions.
- Order by must be followed by Limit n.
- The Select expression does not support subqueries. You can rewrite the code to the Join clause to use subqueries.
- The Join clause does not support Cartesian projects and usage of MapJoin.
- Union all must use the format of subqueries.
- The In/Not in statement can correspond to only one column of subquery, and the number of rows of the returned results cannot exceed 1,000. Otherwise, rewrite the statements using Join.

Compile SQL programs to solve problems

Example 1: List all departments that have at least one employee.

To avoid the sixth point in **Notes for SQL beginners** caused by large data volumes, you must use the Join clause for rewriting. For example:

```
SELECT d.
FROM dept d
JOIN (
    SELECT the DISTINCT deptno AS no
    FROM emp
) e
on d. deptno = e. no;
```

Example 2: List all employees who have higher salaries than Smith.

This example is a typical scenario of MapJoin, as shown in the following code:

```
SELECT /*+ MapJoin(a) */ e.empno
    , e. ename
    , e. sal
FROM emp E
JOIN (
    SELECT MAX(sal) AS sal
    FROM 'emp'
    WHERE `ENAME` = 'SMITH'
) a
ON e. sal> a. sal;
```

Example 3: List the name and the immediate superior's name of all employees.

Use non-equi join, as shown in the following code:

```
SELECT a. ename
    , b. ename
FROM emp A
LEFT OUTER JOIN emp b
ON b. empno = a. mgr;
```

Example 4: List all jobs of which salaries are higher than 1500 yuan.

Use Having, as shown in the following code:

```
SELECT emp. 'JOB'
    , MIN(emp.sal) AS sal
FROM 'emp'
GROUP BY emp. 'JOB'
HAVING MIN(emp.sal) > 1500;
```

Example 5: List the number of employees of each department, and their average salary and average service year.

You can use many built-in functions for time processing, as shown in the following code:

```
SELECT COUNT(empno) AS cnt_emp
    , ROUND(AVG(sal), 2) AS avg_sal
    , ROUND(AVG(datediff(getdate(), hiredate, 'dd')), 2) AS avg_hire
FROM 'emp'
GROUP BY 'DEPTNO';
```

Example 6: List the names of first three employees who have the highest salaries and their ranks (Top n is frequently used).

The SQL statements are as follows:

```
SELECT *
FROM (
  SELECT deptno
    , ename
    , sal
    , ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC) AS
nums
  FROM emp
) emp1
WHERE emp1.nums < 4;
```

Example 7: Use the SQL program to list the number of employees and the ratio of clerks of each department.

The SQL statements are as follows:

```
SELECT deptno
    , COUNT(empno) AS cnt
    , ROUND(SUM(CASE
      WHEN job = 'CLERK' THEN 1
      ELSE 0
    END) / COUNT(empno), 2) AS rate
FROM `EMP`
GROUP BY deptno;
```

9 Basic differences with standard SQL and solutions

This article describes problems that occur frequently when users who are familiar with the relational database SQL use MaxCompute SQL. For specific MaxCompute SQL syntax, see *SQL Overview*.

Basic differences of MaxCompute SQL

Use cases

- Transactions are not supported. (The commit and rollback actions are not supported. We recommend using the codes that have idempotence and support re-running. We also recommend using Insert Overwrite, instead of Insert Into, to write data.)
- Indexes and primary and foreign key constraints are not supported.
- Auto increment fields and default values are not supported. If the default value exists, assign the value when writing the data.

Table partitions

- A table supports 60,000 partitions.
- *The number of input partitions during a query cannot exceed 10,000*. Otherwise, an error is returned. If the partitions are list partitions and the query performs filtering based only on list partitions, an error may be returned if the total number of partitions exceeds 10,000.
- *The number of output partitions in a query cannot exceed 2048*.

Precision

- The Double type has precision. Therefore, we do not recommend that you directly join two Double fields using an equal sign (=). We recommend deducting one number from the other, and consider the two numbers as the same if their difference is smaller than a preset value, for example, abs(a1- a2) < 0.000000001.
- Currently, MaxCompute SQL supports the high-precision type Decimal. If you have higher precision requirements, you can store the data as the string type and use UDF to implement such calculation.

Data type conversion

- If two different field types need to be joined, to prevent unexpected errors, we recommend first converting the types before joining them. This also helps code maintenance.

- For implicit conversions of dates and strings, to input a string to a function that requires entering a date, you can convert between the string and date according to the yyyy-mm-dd hh:mm:ss format. For other formats, see *Date functions > TO_DATE*.

DDL difference and solutions

Table structure

- The partition column name cannot be changed. Only the value corresponding to the partition column can be changed. For the specific differences between partition columns and partitions, see *FAQs*.
- You can add columns but cannot delete columns and modify the data type of columns. For more information, see *SQL FAQs*. If you must delete columns or modify the column data types, see Modify the structure of MaxCompute tables for the most secure procedure.

DML difference and solutions

INSERT

- The most intuitive syntax difference is that Insert into/overwrite is followed by the keyword **Table**.
- Currently, only syntax of Insert Into/Overwrite Table TableName Select supports batch data insert. Syntax of Insert Into/Overwrite Table TableName Values(xxx) does not support the batch data insert. To write a single record of data, see Insert a single record of data using MaxCompute SQL.

UPDATE/DELETE

- Currently, the Update/Delete statements are not supported. If you want to perform the update and delete actions, see *Update and delete data*.

SELECT

- *The number of input tables cannot exceed 16*.
- The Select field in the Group by query is either the group field of Group By or the aggregate function to be used. From the logic perspective, if a non-group column and Group By Key have multiple records of data, the data cannot be displayed without the aggregate function.
- MaxCompute does not support Group by cube (group by with rollup). However, you can use the union clause to simulate Group by cube, for example, `select k1, null, sum(v) from t group by k1 union all select k1, k2, sum(v) from t group by k1, k2;`.

Subquery

Subqueries must have aliases. We recommend that all queries have aliases.

IN/NOT IN

- The data volume in the subquery following In/Not In and Exist/Not Exist cannot exceed 1000. To solve the problem, see *this article*. If the uniqueness of the results returned by the subquery is guaranteed in the business aspect, you can consider removing Distinct to improve the query performance.

10,000 results returned by MaxCompute SQL

- MaxCompute limits the number of data records returned by the separately executed Select clause. For the specific configurations, *see Other operations*. The maximum data number is set to 10,000. If the number of data records to be queried is large, see *this article* to obtain all data while using the Tunnel Command.

MAPJOIN

- The Join clause does not support Cartesian products. The Join clause must use the on clause to set the joining conditions. If you want to create broadcast tables for small tables, you must use MapJoin Hint. For more information, see *this article*.

ORDER BY

- **Order By must be followed by Limit n**. If you want to sort a large number of data records, or even sort the full table, you can set N to a large number. However, use this method with caution. Because the system does not have advantages of distributed systems, the performance may be affected. For more information, see *Sort the queried data in MaxCompute* .

UNION ALL

- The data types, quantities, and names of all columns involved in the UNION ALL calculation must be consistent. Otherwise, an exception is returned.
- The UNION ALL query must be nested by a subquery.