Blackhole SQL引擎使用手册

一、SQL引擎简介

1.功能简介

Blackhole sql是一款面向大数据的、旨在提供对结构化数据使用SQL语句进行查询、分析、统计等功能的单机计算引擎，提供了数据导入\导出能力，和Blackhole的其他两大模块DataFrame和ML能够无缝对接。

2.应用场景

主要应用场景包括如下：

（1）数据存储和查询

（2）业务数据数据统计

（3）业务行为统计和分析

（4）日志分析

（5）商业智能/多维度分析\查询

3.常用SQL分析场景举例

假设存在两份数据events.csv，users.csv，分别记录了用户的访问、下单和购买信息以及用户的个人信息，下面从不同的分析场景举例来说明如何使用SQL引擎进行常用的用户行为分析。

events.csv的数据格式如下：

event_id,user_id,event,time,item_id,fee

1,83,pay_order,2021-04-09 01:13:20,sp_19819,85.01

2,1,pay_order,2021-04-08 09:52:18,sp_55012,50.97

3,72,add_cart,2021-04-06 05:39:10,sp_1044,0.0

4,73,visit,2021-04-07 14:28:30,sp_14826,0.0

5,53,visit,2021-04-10 08:28:59,sp_38361,0.0

6,94,pay_order,2021-04-05 22:19:46,sp_79211,5.19

...

user.csv的数据内容如下：

user_id equip user_name age gender city

1 2 z6s5g0duce 28 1 长春

2 2 t+nhd2scbv 18 1 南京

3 4 4u8dmz+xgh 27 1 长沙

4 1 r4y5ti16j+ 60 1 广州

5 2 ilgt53hb7c 62 2 兰州

...

代码如下：

```
from blackhole.sql.context import Context
from blackhole.sql.dataset import Dataset



schema_events = '''event_id Int64,user_id Int32,event Enum('visit'=1,'add_cart'=2,'pay_order'=3),''' \

'''time DateTime,item_id String,fee Float32'''



schema_users = '''user_id Int32,equip Enum('android'=1,'ios'=2,'wm'=3,'pc'=4),user_name String,age Int8,''' \

'''gender Enum('男'=1,'女'=2),city String'''



format = 'CSV'

context = Context()

table_events = 'events'

table_users = 'users'



# 先清理环境

drop_events_table = 'drop table if exists %s' % table_events

drop_users_table = 'drop table if exists %s' % table_users

context.sql(drop_events_table)

context.sql(drop_users_table)

sql = '''SELECT * FROM file('/home/huhao/data/events.csv', 'CSV', "%s")''' % schema_events

context.sql(sql).show()



# 建表并导入数据
```

```
context.load('/home/huhao/data/events.csv').schema(schema_events).format(format).create_table(table_events)

context.load('/home/huhao/data/users.csv').schema(schema_users).format(format).create_table(table_users)


# 日访问量(PV统计)

sql_pv = '''SELECT count() as "今日PV" FROM events as t

WHERE toDate(t.time)=today() AND t.event='visit' '''

context.sql(sql_pv).show()
```

结果：

今日PV

6

```
# 日活用户量(UV统计)

sql_uv = '''SELECT count(DISTINCT t.user_id) as "今日UV" FROM events as t

WHERE toDate(t.time)=today() AND t.event='visit' '''

context.sql(sql_uv).show()
```

结果：

今日UV

6

```
# 最近7天日活

sql_7days_uv = '''SELECT toString(toDate(t.time)) as "日期", count(DISTINCT t.user_id) as "当日UV" FROM events as t

WHERE t.event='visit' AND toDate(t.time) BETWEEN today()-7 AND today()

GROUP BY toDate(t.time)'''

context.sql(sql_7days_uv).show()
```

结果：

日期 当日UV

2021-04-05 35

2021-04-06 29

2021-04-07 36

2021-04-08 41

2021-04-09 30

2021-04-10 34

2021-04-11 34

2021-04-12 6

```
# 今天分时活跃数
sql_time_uv = '''SELECT toHour(t.time) as "时段", count(DISTINCT t.user_id) as "小时UV" FROM events as t
WHERE t.event='visit' AND toDate(t.time)=today()
GROUP BY toHour(t.time)'''
context.sql(sql_time_uv).show()
```

结果：

时段 小时UV

0 3

1 1

2 2

```
# 查询每天上午 10 点至 11 点的下单用户数
sql_10_11_up = '''SELECT toString(toDate(t.time)) as "日期", count(DISTINCT t.user_id) as "下单用户数" FROM events as t
WHERE t.event='add_cart' AND EXTRACT(HOUR FROM t.time) IN (10,11)
GROUP BY toDate(t.time)'''
context.sql(sql_10_11_up).show()
```

结果：

日期 下单用户数

2021-04-05 11

2021-04-06 3

2021-04-07 6

2021-04-08 5

2021-04-09 5

2021-04-10 3

2021-04-11 4

```
# 查询来自某个城市的用户有多少
sql_city_users = '''SELECT t.city as "城市", count(t.user_id) as "用户数" FROM users as t
GROUP BY t.city'''
context.sql(sql_city_users).show()
```

结果：

城市 用户数

大连 2

苏州 4

天津 3

福州 3

成都 4

厦门 3

长春 3

杭州 1

上海 6

南京 6

```
# 漏斗分析 visit（访问）—add_cart（下单）—pay_order（支付）（窗口期 48 小时且严格满足事件先后顺序
sql_vap_analyze = ''' SELECT count(DISTINCT t.user_id) as "全流程用户数" FROM
(
SELECT user_id, windowFunnel(172800)(time, event='visit',event='add_cart',event='pay_order') as level
FROM events
GROUP BY user_id
) as t WHERE t.level=3 '''
context.sql(sql_vap_analyze).show()
```

结果：

41

```
# 统计连续3(n)天访问的用户数
sql_3days_continues_visit = ''' SELECT count(t2.user_id) as "连续3天访问用户数" FROM
(
SELECT user_id, windowFunnel(3)(t.dt, runningDifference(t.dt)=1, runningDifference(t.dt)=1) as level FROM
(
SELECT user_id, toDate(time) as dt FROM events ORDER BY user_id, time
) as t GROUP BY t.user_id
) as t2 WHERE t2.level=2 '''
context.sql(sql_3days_continues_visit).show()
```

结果：

84

```
# 统计过去3天内浏览最多的3件商品
sql_top_sp_in_3_days = ''' SELECT topK(3)(t.item_id) as res FROM events as t WHERE t.event='visit' AND toDate(now()) - toDate(t.time) <=3
'''
context.sql(sql_top_sp_in_3_days).show()
```

结果：

['sp_6299','sp_64841','sp_34541']

```
# 统计过去3天内消费最多的3位用户
sql_top_user_in_3_days = ''' SELECT t.user_id, t.fees as res FROM
(
SELECT user_id, sum(fee) as fees from events WHERE toDate(now()) - toDate(events.time) <=3 GROUP BY user_id
) as t ORDER BY res DESC limit 3 '''
context.sql(sql_top_user_in_3_days).show()
```

结果：

81 275.3199977874756

7 263.4100036621094

52 249.01000308990479

以上例子几乎涵盖了绝大部分用户行为分析的场景，当然用法不止上面这些，还有一些变通的用法，这里这是举例说明如何使用SQL引擎

二、数据类型定义

数值类型

Int类型

固定长度的整数类型又包括有符号和无符号的整数类型。

- 有符号整数类型

类型	字节	范围
Int8	1	[-2^7 ~2^7-1]
Int16	2	[-2^15 ~ 2^15-1]
Int32	4	[-2^31 ~ 2^31-1]
Int64	8	[-2^63 ~ 2^63-1]
Int128	16	[-2^127 ~ 2^127-1]
Int256	32	[-2^255 ~ 2^255-1]

- 无符号类型

类型	字节	范围
UInt8	1	[0 ~2^8-1]
UInt16	2	[0 ~ 2^16-1]
UInt32	4	[0 ~ 2^32-1]
UInt64	8	[0 ~ 2^64-1]
UInt256	32	[0 ~ 2^256-1]

浮点类型

- 单精度浮点数

Float32从小数点后第8位起会发生数据溢出

类型	字节	精度

Float32	4	7

- 双精度浮点数

Float32从小数点后第17位起会发生数据溢出

类型	字节	精度
Float64	8	16

Decimal类型

有符号的定点数，可在加、减和乘法运算过程中保持精度。此处提供了Decimal32、Decimal64和Decimal128三种精度的定点数，支持几种写法：

Decimal(P, S)

- Decimal32(S)

  数据范围：(-1 * 10^(9 - S), 1 * 10^(9 - S))
- Decimal64(S)

  数据范围：(-1 * 10^(18 - S), 1 * 10^(18 - S))
- Decimal128(S)

  数据范围：(-1 * 10^(38 - S), 1 * 10^(38 - S))
- Decimal256(S)

  数据范围：(-1 * 10^(76 - S), 1 * 10^(76 - S))

其中：**P**代表精度，决定总位数（整数部分+小数部分），取值范围是1～76

S代表规模，决定小数位数，取值范围是0～P

根据**P**的范围，可以有如下的等同写法：

P 取值	原生写法示例	等同于
[1 : 9]	Decimal(9,2)	Decimal32(2)
[10 : 18]	Decimal(18,2)	Decimal64(2)
[19 : 38]	Decimal(38,2)	Decimal128(2)
[39 : 76]	Decimal(76,2)	Decimal256(2)

注意点：不同精度的数据进行四则运算时，**精度(总位数)和规模(小数点位数)**会发生变化，具体规则如下：

- 精度对应的规则
  - Decimal64(S1) Decimal32(S2) -> Decimal64(S)
  - Decimal128(S1) Decimal32(S2) -> Decimal128(S)
  - Decimal128(S1) Decimal64(S2) -> Decimal128(S)
  - Decimal256(S1) Decimal<32|64|128>(S2) -> Decimal256(S)

  可以看出：两个不同精度的数据进行四则运算时，结果数据以最大精度为准
- 规模(小数点位数)对应的规则
  - 加法|减法：S = max(S1, S2)，即以两个数据中小数点位数最多的为准
  - 乘法： S = S1 + S2(注意：S1精度 >= S2精度)，即以两个数据的小数位相加为准
- 除法： S = S1，即被除数的小数位为准

字符串类型

String

字符串可以是任意长度的。它可以包含任意的字节集，包含空字节。因此，字符串类型可以代替其他 DBMSs 中的VARCHAR、BLOB、CLOB 等类型。

FixedString

固定长度的N字节字符串，一般在在一些明确字符串长度的场景下使用，声明方式如下：

```
<column_name> FixedString(N)
-- N
```

注意：FixedString使用null字节填充末尾字符。

日期类型

时间类型分为DateTime、DateTime64和Date三类。需要注意的是目前没有时间戳类型，也就是说，时间类型最高的精度是秒，所以如果需要处理毫秒、微秒精度的时间，则只能借助UInt类型实现。

Date类型

用两个字节存储，表示从 1970-01-01 (无符号) 到当前的日期值。日期中没有存储时区信息。

DateTime类型

用四个字节（无符号的）存储 Unix 时间戳，允许存储与日期类型相同的范围内的值，最小值为 0000-00-00 00:00:00。时间戳类型值精确到秒（没有闰秒），时区使用启动客户端或服务器时的系统时区。

布尔类型

当前版本没有单独的类型来存储布尔值。可以使用UInt8 类型，取值限制为0或 1。

数组类型

Array(T)，由 T 类型元素组成的数组。T 可以是任意类型，包含数组类型。但不推荐使用多维数组，目前对多维数组的支持有限。

```
SELECT array(1, 2) AS x, toTypeName(x);
--
xtoTypeName(array(1, 2))
 [1,2]  Array(UInt8)

SELECT [1, 2] AS x, toTypeName(x);
--
xtoTypeName([1, 2])
 [1,2]  Array(UInt8)
```

```
NullNullable
```

```
SELECT array(1, 2, NULL) AS x, toTypeName(x);
--
xtoTypeName(array(1, 2, NULL))
 [1,2,NULL]  Array(Nullable(UInt8))
```

```
SELECT array(1, 'a')
--
DB::Exception: There is no supertype for types UInt8, String because some of them are String/FixedString anc
 some of them are not
```

枚举类型

枚举类型通常在定义常量时使用，当前版本提供了Enum8和Enum16两种枚举类型。

```
--
CREATE TABLE t_enum
(
    x Enum8('hello' = 1, 'world' = 2)
);
-- INSERT
INSERT INTO t_enum VALUES ('hello'), ('world'), ('hello');
--
INSERT INTO t_enum values('a')
-- Unknown element 'a' for type Enum8('hello' = 1, 'world' = 2)
```

Tuple类型

Tuple(T1, T2, ...)，元组，与Array不同的是，Tuple中每个元素都有单独的类型，不能在表中存储元组（除了内存表）。它们可以用于临时列分组。在查询中，IN表达式和带特定参数的 lambda 函数可以来对临时列进行分组。

```
SELECT tuple(1,'a') AS x, toTypeName(x);
--
xtoTypeName(tuple(1, 'a'))
 (1,'a')  Tuple(UInt8, String)

--
CREATE TABLE t_tuple(
   c1 Tuple(String,Int8)
);
-- INSERT
INSERT INTO t_tuple VALUES(('jack',20));
--
SELECT * FROM t_tuple;
c1
 ('jack',20)

--
INSERT INTO t_tuple VALUES(('tom','20'));
-- Type mismatch in IN or VALUES section. Expected: Int8. Got: String
```

特殊数据类型

Nullable

Nullable类型表示某个基础数据类型可以是Null值。其具体用法如下所示：

```
--
CREATE TABLE t_null(x Int8, y Nullable(Int8));
--
INSERT INTO t_null VALUES (1, NULL), (2, 3);
SELECT x + y FROM t_null;
--
plus(x, y)

          5
```

三、数据导入

1.从外部文件load数据到数据表

SQL引擎支持将外部数据导入到本地数据表中，格式包括CSV、CSVWithNames和Parquet，目前只有Python接口，用法如下：

```
from blackhole.sql.context import Context



data_path = './test_data.csv'

table_name = 'test_table'


ctx = Context()

ds = ctx.load(data_path).format('CSV').schema('id Int64, name String').create_table(table_name)
```

上述代码会将本地的test_data.csv文件数据导入到test_table表中，其中schema为'id Int64, name String'。注意：指定的schema必须与数据源的

schema相同。

2.使用INSERT方式插入数据

1）单行插入

```
from blackhole.sql.context import Context
from blackhole.sql.dataset import Dataset


ctx = Context()
schema = 'id Int64, name String, price Float32'
table_name = 'test_insert_table'
ds = Dataset().set_context(ctx).schema(schema).format('TABLE').create_table(table_name)

insert_sql = '''INSERT INTO %s VALUES(%d,'%s',%f)''' % (table_name, 0, 'apple', 12.37)
ds.sql(insert_sql)
```

2）多行插入

```
from blackhole.sql.context import Context
from blackhole.sql.dataset import Dataset


ctx = Context()
schema = 'id Int64, name String, price Float32'
table_name = 'test_insert_table'
ds = Dataset().set_context(ctx).schema(schema).format('TABLE').create_table(table_name)



insert_sql = 'INSERT INTO %s VALUES' % table_name
for i in range(1, 100):
    insert_sql += '''(%d,'%s',%f),''' % (i, 'apple', 12.37)

insert_sql = insert_sql.strip(',')
ds.sql(insert_sql)
```

注意：String类型字段值的引号不能省略。

四、数据查询

1.从数据表查询

```
from blackhole.sql.context import Context
from blackhole.sql.dataset import Dataset



ctx = Context()

ds = Dataset().set_context(ctx)

query_sql = 'select * from test_table'

result_data = ds.sql(query_sql).raw_data()
```

2.从外部文件查询

```
from blackhole.sql.context import Context
from blackhole.sql.dataset import Dataset



data_path = './test_data.csv'


format = 'CSV'


schema = 'id Int64, name String, price Float32'

ctx = Context()

query_sql = 'select * from file'

result_data = ctx.query_from_file(query_sql, file=data_path, format=format, schema=schema).raw_data()
```

3.查询子句

DISTINCT

查询结果集在指定字段上只保留一行

FROM

FROM 子句指定从以下数据源中读取数据:

- 表
- 子查询
- 表函数

GROUP BY

GROUP BY 子句将 SELECT 查询结果转换为聚合模式，目前支持的聚合函数如下：

sum–对指定字段进行求和，只能对数值型字段求和

min–求指定字段在查询结果集中的最小值

max–求指定字段在查询结果集中的最大值

HAVING

允许过滤由 GROUP BY 生成的聚合结果. 它类似于 WHERE ，但不同的是 WHERE 在聚合之前执行，而 HAVING 在之后进行。

INTO OUTFILE

SELECT query 将其输出重定向到本机上的指定文件。

JOIN

连接两个表

支持的联接类型

所有标准 SQL JOIN 支持类型:

- `INNER JOIN`，只返回匹配的行。
- `LEFT OUTER JOIN`，除了匹配的行之外，还返回左表中的非匹配行。
- `RIGHT OUTER JOIN`，除了匹配的行之外，还返回右表中的非匹配行。
- `FULL OUTER JOIN`，除了匹配的行之外，还会返回两个表中的非匹配行。
- `CROSS JOIN`，产生整个表的笛卡尔积，“join keys” 是 不 指定。

`JOIN` 没有指定类型暗指 `INNER`. 关键字 `OUTER` 可以安全地省略。

LIMIT

`LIMIT m` 允许选择结果中起始的 `m` 行。

ORDER BY

`ORDER BY` 子句包含一个表达式列表，每个表达式都可以用 `DESC` （降序）或 `ASC` （升序）修饰符确定排序方向。 如果未指定方向, 默认是 `ASC`

SAMPLE

SAMPLE K

这里 `k` 从0到1的数字（支持小数和小数表示法）。 例如, `SAMPLE 1/2` 或 `SAMPLE 0.5k`

示例：

SELECT

Title,

count() * 10 AS PageViews

FROM hits_distributed

SAMPLE 0.1

WHERE

CounterID = 34

GROUP BY Title

ORDER BY PageViews DESC LIMIT 1000

SAMPLE N

这里 n 是足够大的整数。 例如, `SAMPLE 10000000`.

在这种情况下，查询在至少 n 行（但不超过这个）上执行采样。 例如, `SAMPLE 10000000` 在至少10,000,000行上执行采样。

UNION ALL

联合两个子查询的结果

示例：

SELECT CounterID, 1 AS table, toInt64(count()) AS c

FROM test.hits

GROUP BY CounterID

UNION ALL

SELECT CounterID, 2 AS table, sum(Sign) AS c

FROM test.visits

GROUP BY CounterID

HAVING c > 0

WHERE

指定条件对查询结果集进行过滤，支持OR，AND等条件组合。

五、数据导出

1.直接展示

示例：

```
from blackhole.sql.context import Context
ctx = Context()
query_sql = 'select * from test_table'
```

ctx.sql(query_sql).show()

默认打印前10行

2.导出到文件

示例：

from blackhole.sql.context import Context

```
ctx = Context()
query_sql = 'select * from test_table'
```

ctx.sql(query_sql).save('/home/data/test.tsv')

默认为tsv格式文件，即使用tab做为字段分隔符。

六、函数

1.普通函数

2.聚合函数