

Greenplum 应用开发

目录

Greenplum 应用开发.....	1
目录.....	1
1 GP 数据库设计.....	2
1.1 逻辑数据模型.....	2
1.1.1 支持通用的数据建模方式.....	2
1.1.2 Greenplum 支持.....	2
1.1.3 Greenplum 不支持.....	3
1.1.4 星型模型概述.....	3
1.1.4.1 事实表(Fact).....	3
1.1.4.2 维表(Dimension).....	3
1.1.4.3 星型模型实例.....	3
1.1.5 雪花模型实例.....	4
1.1.6 键和唯一约束.....	5
1.1.7 属性取值范围约束.....	5
1.1.8 参照完整性约束(外键).....	6
1.1.9 逻辑数据模型设计小结.....	6
1.2 物理数据模型.....	7
1.2.1 Greenplum 高性能之关键.....	7
1.2.2 数据分布 Sharding.....	7
1.2.3 分布键(Distribution key)分布过程.....	8
1.2.4 分布键与查询性能.....	10
1.2.4.1 最优查询性能.....	10
1.2.4.2 数据重分布查询.....	11
1.2.4.3 分布键与查询性能总结.....	11
1.2.5 数据倾斜.....	12
1.2.6 分布键的选择原则总结.....	12
1.3 Greenplum 储存引擎.....	13
1.3.1 HEAD 堆表.....	13
1.3.2 AO 追加优化表.....	13
1.3.3 外部表.....	14
1.4 Greenplum 压缩.....	14
1.4.1 支持的压缩算法.....	14
1.4.2 压缩算法的性能对比.....	15
1.5 表分区 Partitioning.....	15
1.5.1 表分区概念.....	15
1.5.2 分区裁剪.....	16

1.6 表分区设计.....	17
1.6.1 表分区设计原则.....	17
1.6.2 表分区实例.....	17
1.7 索引.....	18
1.8 多态储存.....	18
1.9 物理数据模型设计小结.....	19
2 GP 的应用开发.....	19
2.1 SQL 和事务.....	19
2.1.1 事务与应用的实例.....	19
2.1.2 事务的总结.....	21
2.2 时间日期和内置函数.....	21
2.2.1 注意查看区别.....	21
2.2.2 常用的日期与实践 SQL.....	22
2.2.3 时区避坑技巧.....	22
2.3 UDF 和 UDT.....	23
2.3.1 内置的 UDF.....	23
2.3.2 常见的 UDF 语言.....	23
2.4 External Web Table.....	24
2.5 公用表表达式 CTE.....	24
2.6 ROLLUP,CUBE,and GROUPING SETS.....	27
2.7 窗口函数.....	27
2.8 GP 扩展介绍.....	27

1 GP 数据库设计

1.1 逻辑数据模型

1.1.1 支持通用的数据建模方式

- 1、星型模型
- 2、雪花模型
- 3、3NF,BCNF,4NF

1.1.2 Greenplum 支持

- 1、主键(Primary key)
- 2、单值约束(UNIQUE)
- 3、属性取值范围约束(空值，CHECK)

1.1.3 Greenplum 不支持

- 1、参照完整性约束(Foreign Key)

1.1.4 星型模型概述

1.1.4.1 事实表(Fact)

- 1、存放大量事实业务交易，事件
- 2、分析作业对事实表进行聚集
- 3、属性可以非常多，允许冗余
- 4、数据量不断增长，周期性通过 ETL 作业载入新数据
- 5、通常很少发生数据变更

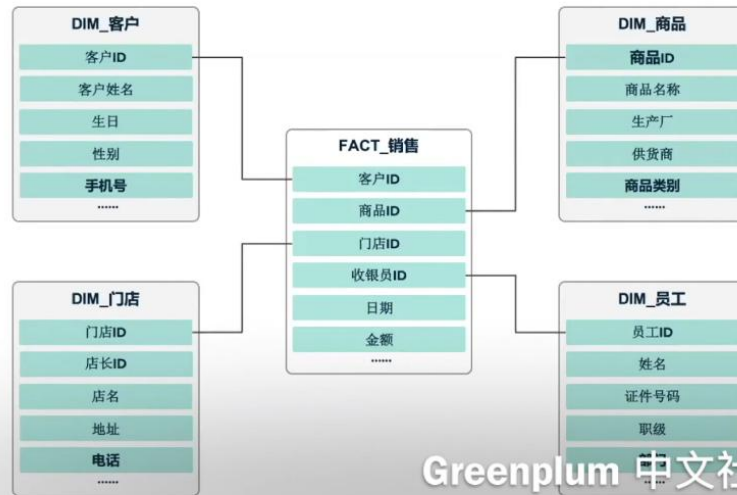
1.1.4.2 维表(Dimension)

- 1、为事实表中一项或多项属性提供额外的描述信息
- 2、相对于事实表，数据量非常小
- 3、特定条件下会发生更新
- 4、一个事实表通常有多个维度表，构型星型模型

1.1.4.3 星型模型实例

FACT_销售表示事实表，其余的表都属于维表

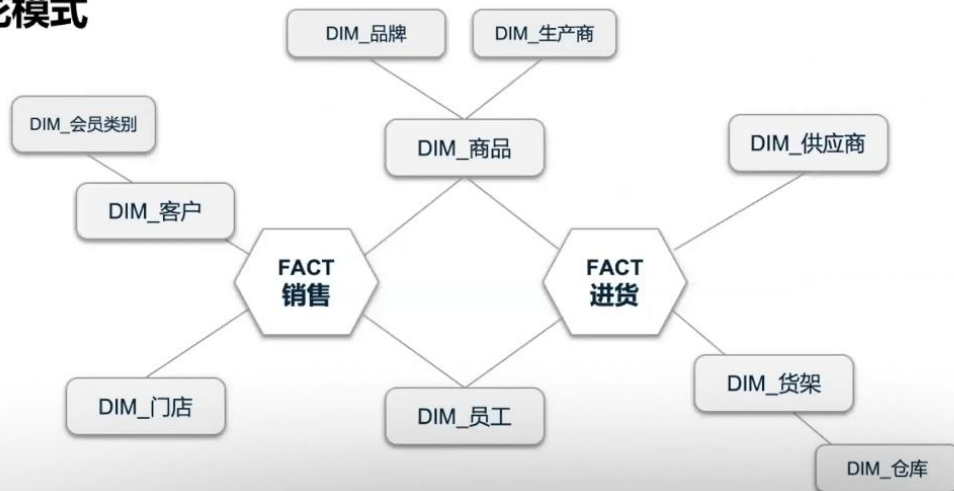
星型模式



1.1.5 雪花模型实例

多个星型模型表就组成了雪花模型

雪花模式



1.1.6 键和唯一约束

segmentfault 问答

键和唯一约束

- Greenplum支持使用Primary Key来唯一标识一个元组
- 支持使用自增数据类型

```
CREATE TABLE serial_demo (id SERIAL PRIMARY KEY);
```

- 主键和唯一约束的前序子集 = 分布键 (GP4/5)
主键和唯一约束的任意子集 = 分布键 (GP6)

```
CREATE TABLE unique_demo (  
  age INT, name TEXT, town TEXT, event VARCHAR(64),  
  UNIQUE (name, town, event)  
) DISTRIBUTED BY (name, town);
```

- 指定主键或唯一约束会自动创建索引
- 一个表可以包含多个唯一约束 (GP6)
- OLAP应用对比OLTP, 较少用到主键或唯一约束

Greenplum 中文社区所有

1.1.7 属性取值范围约束

如果字段设置了范围约束, 而查询时超过了范围 GP 优化器会认为条件不成立, 就没有约束, 这样会对查询的性能有极大的影响。

segmentfault 问答

属性取值范围约束

```
CREATE TABLE check_demo (  
  id INT NOT NULL  
  , name TEXT CHECK (name IN ('Zhao', 'Qian', 'Sun', 'Li'))  
  , age INT CHECK (age BETWEEN 0 AND 120)  
  , event TEXT CHECK (event NOT LIKE 'dance%')  
  , val1 INT  
  , val2 INT  
  , CHECK (val1 + val2 < 100)  
);
```

值域约束会被优化器用于简化查询计划

```
EXPLAIN SELECT * FROM unique_demo, check_demo WHERE age = 200;  
QUERY PLAN  
  
-----  
Result (cost=0.00..0.01 rows=1 width=0)  
One-Time Filter: false  
Optimizer: Postgres query optimizer
```

Greenplum 中文社区所有

1.1.8 参照完整性约束(外键)

segmentfault 课堂

参照完整性约束（外键）

- Greenplum能够接受包含外键约束的DDL语句
- 但是不会在插入数据时实际进行外键检查
- 使用者需要在数据来源层面确保参照完整性

```
CREATE TABLE fk_demo (  
    id INT  
    , f_id INT REFERENCES pk_demo (id)  
) DISTRIBUTED BY (id);  
WARNING: referential integrity (FOREIGN KEY) constraints are  
not supported in Greenplum Database, will not be enforced  
CREATE TABLE
```

Greenplum 中文社区所有

1.1.9 逻辑数据模型设计小结

segmentfault 课堂

逻辑数据模型设计小结

优先考虑业务需求和便于使用

- 星型模式和雪花模式相当自由，没有太多一定之规的限制
- 基于业务决定需要哪些事实表和维表
- 确定事实表和维表中哪些列和冗余
- 不要过分范式化，会带来大量小表连接导致难以使用
- 创建很宽的扁平化事实表是完全合理的
- 数据模型的开发也是不断迭代改良的过程
- 利用数据库系统本身验证新的数据模型

Greenplum 中文社区所有

1.2 物理数据模型

1.2.1 Greenplum 高性能之关键

- 1、分布键、随机分布
- 2、分布键的选择
- 3、储存引擎 Heap,AO,AOCO 和外部表
- 4、压缩
- 5、表分区
- 6、索引
- 7、多态分布

1.2.2 数据分布 Sharding

数据分布 Sharding

segmentfault 社区

数据Segment之间分布，是Shared-Nothing MPP数据库的核心特性
正确使用会带来理想的性能和扩展性
不佳的分布方式会加重数据库的负担，降低效能

Greenplum支持的数据分布方式

- 通过分布键的值进行散列
- 随机分布

Greenplum 中文社区所有

数据分布 Sharding

分布键

- 创建表时指定某个或多个列为分布键DISTRIBUTION KEY
- 插入数据时，GP对被插入数据的DISTRIBUTION KEY进行哈希，映射到某个Segment上
- NULL值与其他值一样，也会被哈希到一个Segment上

随机分布

- 插入数据时，会随机插入到某个Segment上
- GP会保证数据会平均分布在各个Segment之间
- 使用场景有限，通常作为ETL的临时表

Greenplum 中文社区所有

1.2.3 分布键(Distribution key)分布过程

分布键 DISTRIBUTION KEY

```
CREATE TABLE dk_demo (  
  val1 INT,  
  val2 TEXT  
) DISTRIBUTED BY (val1);
```

(25, 'Zhao')
(20, 'Li')
(10, 'Wang')



DK = val1
val1 = 10
HASH(10) = 1



Greenplum 中文社区所有

分布键 DISTRIBUTION KEY

```
CREATE TABLE dk_demo (  
  val1 INT,  
  val2 TEXT  
) DISTRIBUTED BY (val1);
```

(25, 'Zhao')

(20, 'Li')



DK = val1
val1 = 10
HASH(10) = 1



Greenplum 中文社区所有

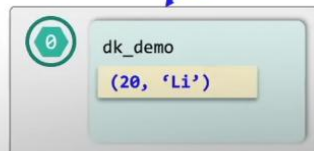
分布键 DISTRIBUTION KEY

```
CREATE TABLE dk_demo (  
  val1 INT,  
  val2 TEXT  
) DISTRIBUTED BY (val1);
```

(25, 'Zhao')



DK = val1
val1 = 20
HASH(20) = 0

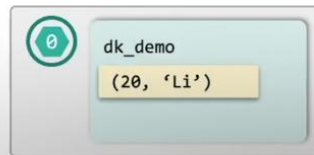


Greenplum 中文社区所有

分布键 DISTRIBUTION KEY

```
CREATE TABLE dk_demo (
  val1 INT,
  val2 TEXT
) DISTRIBUTED BY (val1);
```

Master
DK = val1
val1 = 25
HASH(25) = 1



Greenplum 中文社区所有

1.2.4 分布键与查询性能

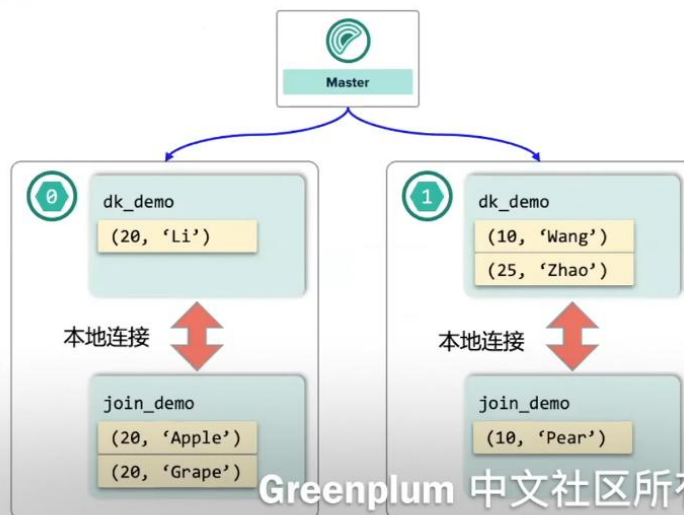
1.2.4.1 最优查询性能

分布键与查询性能

```
CREATE TABLE dk_demo (
  val1 INT,
  val2 TEXT
) DISTRIBUTED BY (val1);
```

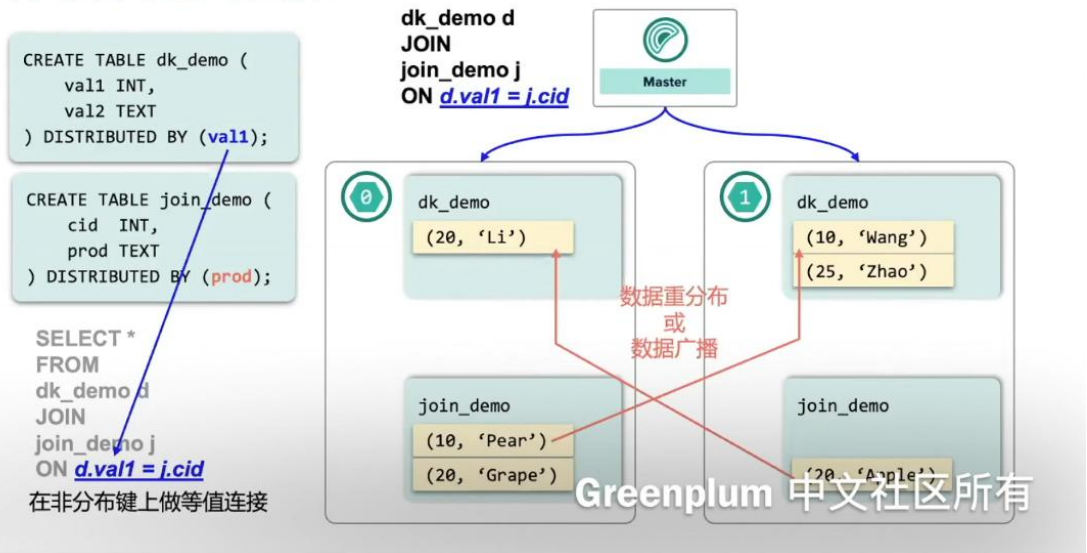
```
CREATE TABLE join_demo (
  cid INT,
  prod TEXT
) DISTRIBUTED BY (cid);
```

```
SELECT *
FROM
  dk_demo d
JOIN
  join_demo j
ON d.val1 = j.cid
在分布键上做等值连接
```



1.2.4.2 数据重分布查询

分布键与查询性能



1.2.4.3 分布键与查询性能总结

分布键与查询性能

- 本地连接 > 单个表重分布或广播 > 多个表重分布或广播
- 重分布或广播小表 > 重分布或广播大表（优化器决定）
- 随机分布的表，参与任何查询时都会引起重分布或广播，因此查询性能不佳

- 例，两个表各500万行
 - 本地连接 = 3秒
 - 一个表重分布 = 4秒
 - 两个表都是随机分布 = 16秒

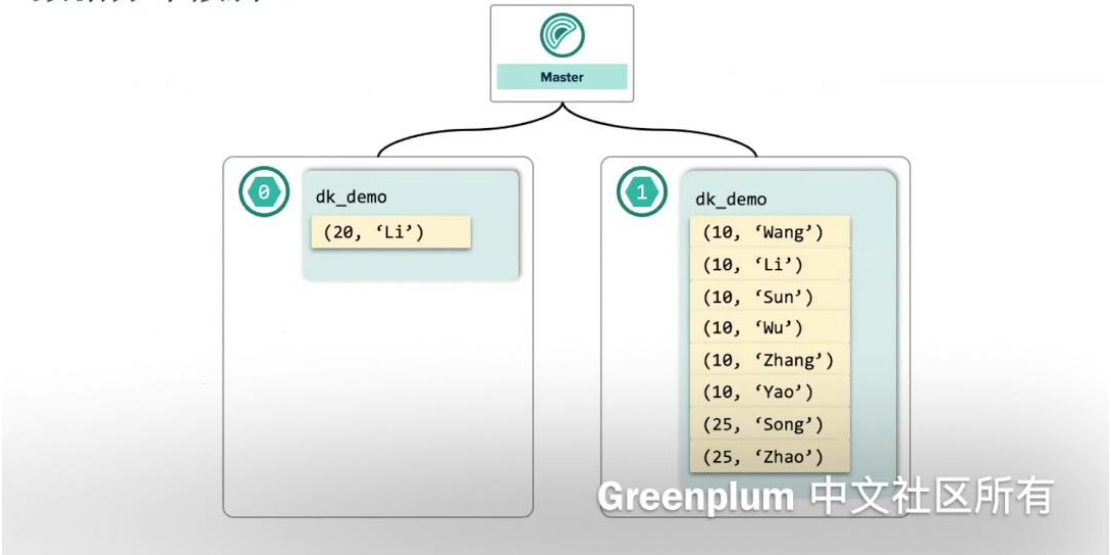
实际差异与Segment个数及硬件、网络条件有关，本结果仅供参考

Greenplum 中文社区所有

1.2.5 数据倾斜

数据分布倾斜

segmentfault 课堂



1.2.6 分布键的选择原则总结

segmentfault 课堂

分布键的选择原则

- 唯一值足够多的列
- 该列上特定的值（尤其是NULL）不会明显多于其他取值
- 考虑大表之间频繁做等值连接的列
- 不经常发生更新，最好是从不更新的列

	唯一值个数	每个值最大行数	是否能够作为分布键
column 1	3457859	1	可
column 2	5000	891	可
column 3	5	700000	
column 4	20000	3235754	

Greenplum 中文社区所有

1.3 Greenplum 储存引擎

1.3.1 HEAD 堆表

存储引擎

segmentfault 课堂

HEAP 堆表	追加优化表	外部表
<ul style="list-style-type: none">● 源自PostgreSQL，是GP的默认存储引擎● 通用型最佳，对于OLTP场景下的INSERT、UPDATE、DELETE提供完整的事务隔离级别支持。● 适合数据量较小但需要更新、删除的表，例如维表		

Greenplum 中文社区所有

1.3.2 AO 追加优化表

存储引擎

segmentfault 课堂

HEAP 堆表	追加优化表	外部表
<ul style="list-style-type: none">● 支持行存AO和列存AOCO格式● 列存储AOCO是最适合OLAP的存储引擎● 大部分用户首选AOCO作为事实表● 支持事务，但在可串行化隔离级别下的UPDATE和DELETE操作不如HEAP 不建议频繁更新或删除● 列式存储支持专用的压缩算法		

```
CREATE TABLE aoco_demo (  
    .....  
)  
WITH (  
    APPENDONLY = true  
    , ORIENTATION = column  
    , COMPRESSTYPE =  
    RLE_TYPE  
    , COMPRESSLEVEL = 2  
    DISTRIBUTED BY (...));
```

Greenplum 中文社区所有

1.3.3 外部表

segmentfault 课堂

存储引擎

HEAP 堆表

追加优化表

外部表

- 通常用于数据ETL的中间过程或廉价存储年代久远的历史数据
- 不享受GP内部的事务和高可用支持
- 包括可读和可写的外部表
- 常见外部表有 文件、S3对象存储、HDFS等

扩展资料

GP外部表全方位介绍
<https://greenplum.cn/2019/06/14/greenplum-external-table/>

Greenplum 中文社区所有

1.4 Greenplum 压缩

1.4.1 支持的压缩算法

segmentfault 课堂

压缩

追加优化存储AO/AOCO可以压缩数据

压缩算法	说明	可选的压缩级别
RLE_TYPE	列存储专用	1-4
ZLib	GP5常规压缩	1-9
QuickLZ	仅在商业版Greenplum支持	1
ZStd	GP6新特性	1-19

- 压缩数据可以减少磁盘占用，降低查询的磁盘I/O开销
- 压缩算法的选择需要考虑压缩/解压缩的CPU消耗、速度快慢、以及压缩比高低
- 以上指标与硬件、数据特征、查询调优设置有关，需要使用实际数据在真实环境下测试以找到最佳选择

Greenplum 中文社区所有

1.4.2 压缩算法的性能对比

详细信息请查看:

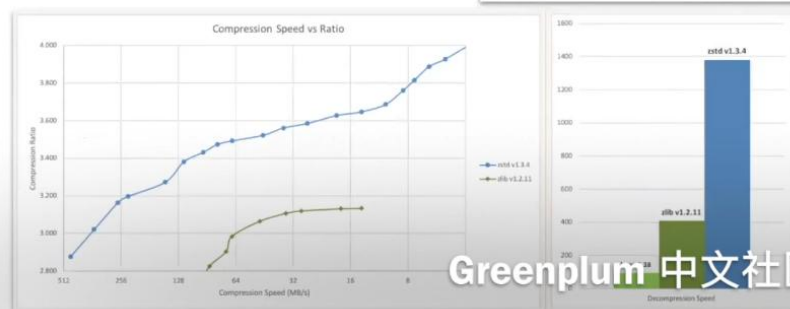
<https://github.com/facebook/zstd>

压缩算法的性能对比

第三方数据仅供参考

来源: <https://facebook.github.io/zstd/>

Compressor name	Ratio	Compression	Decompress.
zstd 1.3.4-1	2.877	470 MB/s	1380 MB/s
zlib 1.2.11-1	2.743	110 MB/s	400 MB/s
brotli 1.0.2-0	2.701	410 MB/s	430 MB/s
quicklz 1.5.0-1	2.238	550 MB/s	710 MB/s
lzo1x 2.09-1	2.108	650 MB/s	830 MB/s
lz4 1.8.1	2.101	750 MB/s	3700 MB/s
snappy 1.1.4	2.091	530 MB/s	1800 MB/s
lzf 3.6-1	2.077	400 MB/s	860 MB/s



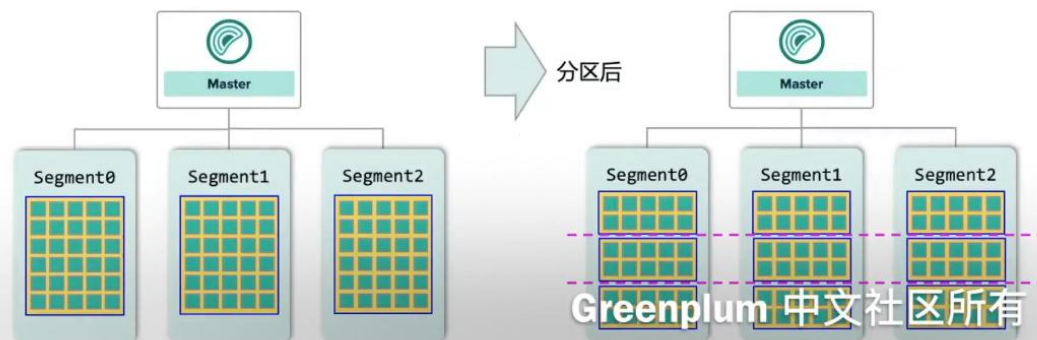
Greenplum 中文社区所有

1.5 表分区 Partitioning

1.5.1 表分区概念

表分区 Partitioning

- 对于数据量更大的事实表，Greenplum建议对表进行分区
- 数据分布（Sharding）将数据分散到各个Segment上
- 表分区（Partitioning）进一步将大表的数据进行水平分片

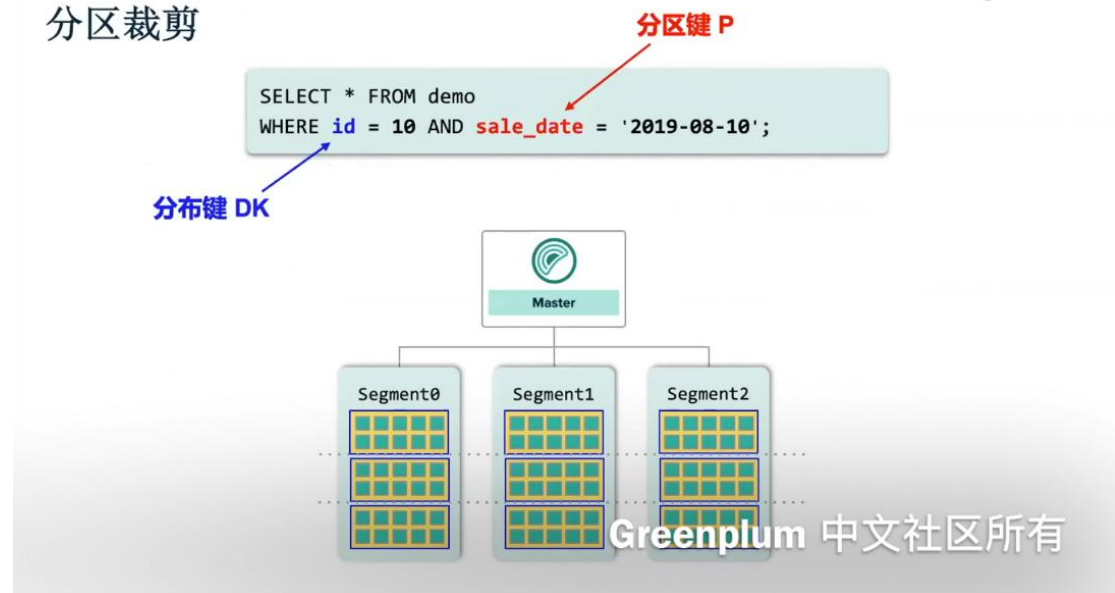


Greenplum 中文社区所有

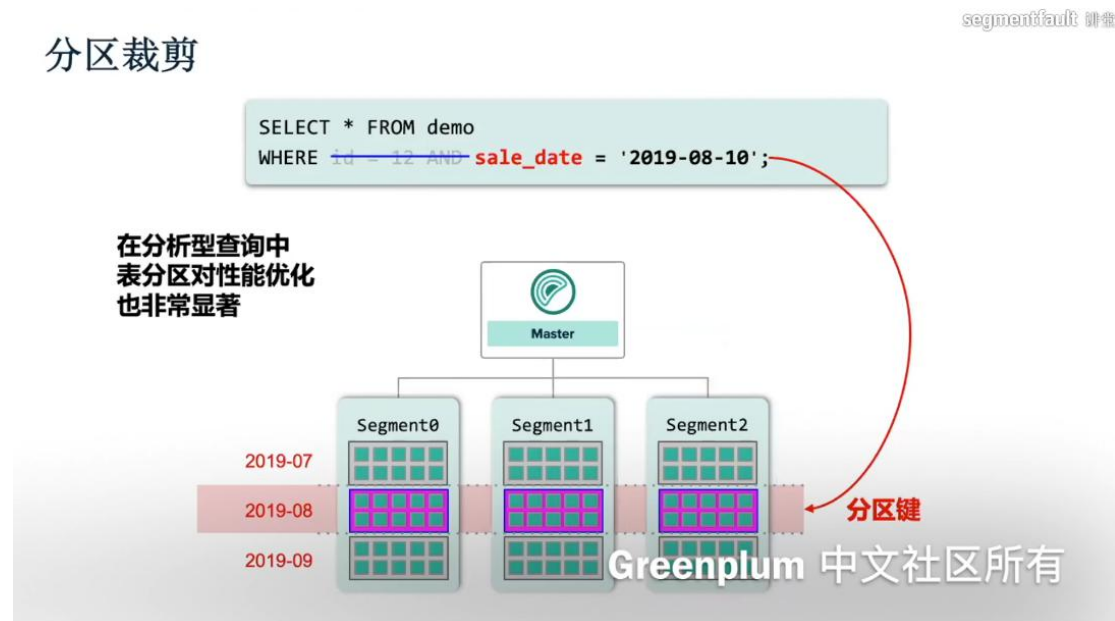
1.5.2 分区裁剪

使用分区裁剪能够快速查询到需要得到的数据

分区裁剪



分区裁剪



1.6 表分区设计

1.6.1 表分区设计原则

segmentfault 课堂

表分区设计原则

何时需要表分区

- 数据量大且持续增长，通常是事实表
- 查询性能出现退化，且已尝试其他常规的优化方式
- 具备线性可扩展的分区窗口，例如：交易日期时间
- 查询条件包含候选的分区键

其他设计考量

- 各个分区的数据量尽量均衡
- 避免滥用多级分区，过多的分区会加重系统负担
- GP支持缺省分区，由于缺省分区不能参与裁剪，尽量避免使用
如必须使用，一定注意缺省分区的数据量要尽量小

Greenplum 中文社区所有

1.6.2 表分区实例

segmentfault 课堂

表分区实例

单级分区

```
CREATE TABLE sales (  
    id int, date date, amt decimal(10,2)  
) DISTRIBUTED BY (id)  
PARTITION BY RANGE (date)  
( START (date '2016-01-01') INCLUSIVE  
  END  (date '2017-01-01') EXCLUSIVE  
  EVERY (INTERVAL '1 month') );
```

多级分区

```
CREATE TABLE p3_sales (  
    id int, year int, month int, day int,  
    region text)  
DISTRIBUTED BY (id)  
PARTITION BY RANGE (year)  
  SUBPARTITION BY RANGE (month)  
    SUBPARTITION TEMPLATE (  
      START (1) END (13) EVERY (1),  
      DEFAULT SUBPARTITION other_months )  
    SUBPARTITION BY LIST (region)  
      SUBPARTITION TEMPLATE (  
        SUBPARTITION usa VALUES ('usa'),  
        SUBPARTITION europe VALUES ('europe'),  
        SUBPARTITION asia VALUES ('asia'),  
        DEFAULT SUBPARTITION other_regions )  
  ( START (2002) END (2012) EVERY (1),  
    DEFAULT PARTITION outlying_years );
```

缺省分区

Greenplum 中文社区所有

1.7 索引

索引不是越创建越好，要通过查看 EXPLAIN 查看执行计划，使用使用了 INDEX SCAN

索引

segmentfault 课堂

在OLAP场景下，索引应该被保守地使用

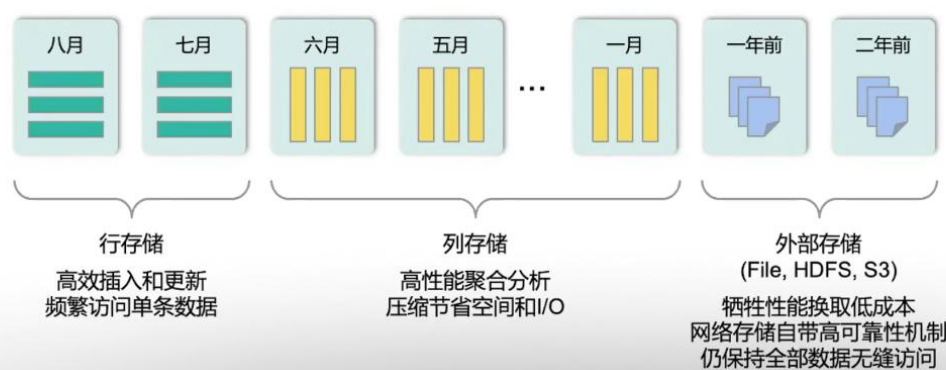
- 一般事务型数据库中，索引可以快速访问指定记录
- GP提供了数据分布和表分区能力，在对全体或部分数据进行连续扫描、聚集的分析型查询中，索引很少有机会带来性能改善
- 索引需要额外空间和维护成本，影响数据ETL速度
- 是否能值得建立索引，最可靠的方法是通过EXPLAIN或者GPCC查看查询计划，确认查询计划能够使用INDEX SCAN，并且查询速度优于没有索引，则证明索引有效

Greenplum 中文社区所有

1.8 多态储存

多态存储

segmentfault 课堂



Greenplum 中文社区所有

1.9 物理数据模型设计小结

物理数据模型设计小结

segmentfault 课堂

- 合理选择分布键，仅在特定场景使用随机分布
- 事实表通常使用AOCO存储，维表可以用HEAP
- 事实表多考虑进行分区，分区策略应简洁有效，避免过多分区
- 仅在确定的有效的前提下创建索引，宁缺毋滥
- 使用EXPLAIN或GPCC确认分布、分区、索引的合理有效型
- 高级用户考虑对核心事实表建立多态存储

Greenplum 中文社区所有

2 GP 的应用开发

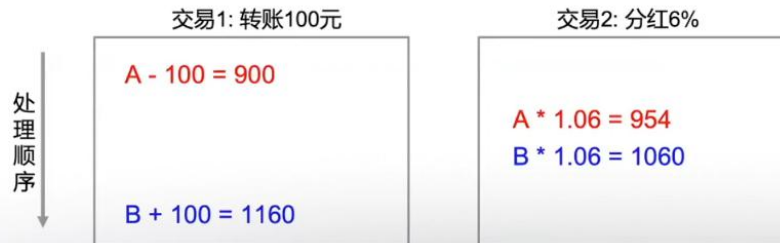
2.1 SQL 和事务

在 greenplum 中使用 BEGIN 开始事务，使用 COMMIT 提交事务

2.1.1 事务与应用的实例

事务与应用

- A和B两人各有1000元
- A给B转账100元
- 同时有一笔6%（共120元）的分红到账，需要分配给A和B



结果 A有954元, B有1160元 共计2114元, 有6元不翼而飞

Greenplum 中文社区所有

事务与应用

- A有100元
- 一笔交易支出了10元，但因其他原因交易提交失败了
- 另一笔交易支出了20元，成功提交



最后 A 余额 70 元



Greenplum 中文社区所有

2.1.2 事务的总结

segmentfault 课堂

事务

以上两种情况，GP都不会中招！

- GP与PostgreSQL一脉相承，“教科书式的”SQL语法和事务支持
- 在分布式OLAP数据库领域，完备的SQL和事务支持产品凤毛麟角
- “教科书式”意味着良好的普适性和拓展性，不会被轻易取代
- 背后多年的积累，巨大的工程量和实现难度

Greenplum 中文社区所有

2.2 时间日期和内置函数

2.2.1 注意查看区别

segmentfault 课堂

日期与时间

● 获取当前时间

```
SELECT now();
SELECT CURRENT_TIMESTAMP;
```

now

2019-08-11 22:28:16.789104+08 含时区

● 获取当前日期

```
SELECT CURRENT_DATE;
```

date

2019-08-11 不含时区

● 获取当前时区

```
SHOW timezone;
```

TimeZone

PRC

⚠ 当分区键 (Partition By) 和查询的 WHERE 条件是否含有时区不一致时，Planner 不会应用分区裁剪。Orca 优化器则不存在此限制。

Greenplum 中文社区所有

2.2.2 常用的日期与实践 SQL

segmentfault 课堂

日期与时间

● 时区转化

```
SELECT now()::TIMESTAMP WITHOUT TIME ZONE;  
now  
-----  
2019-08-11 22:57:16.749054
```

```
SELECT now() AT TIME ZONE 'PRC';  
timezone  
-----  
2019-08-11 22:58:42.298589
```

```
SELECT now() AT TIME ZONE 'UTC';  
timezone  
-----  
2019-08-11 15:00:50.198354
```

● 时间计算

```
SELECT extract(epoch FROM now());  
date_part  
-----  
1565535958.80443
```

```
SELECT now() - INTERVAL '2 weeks';  
?column?  
-----  
2019-07-28 23:25:31.941516+08
```

```
SELECT now() - '2019-01-01';  
?column?  
-----  
222 days 23:32:14.443273
```

Greenplum 中文社区所有

2.2.3 时区避坑技巧

segmentfault 课堂

时区避坑技巧

- 中国标准时区 (CST) 和 美国中部时区 (CST) 重名
- GP默认会将CST识别为美国中部时区
- 导致国内时区为CST的服务器在时间计算时出现意外结果
- 解决方法
 - 修改 GP安装目录/share/postgresql/timezonesets/Default
 - 找到 CST -21600 这行, 修改为 CST 28800
 - 所有 Segment 和 Master 服务器全部修改
 - 重启 GPDB

Greenplum 中文社区所有

2.3 UDF 和 UDT

2.3.1 内置的 UDF

UDF 可以使用循环结构和变量,也可以支持判断和递归

UDF - PL/pgSQL

● 循环结构和变量

```
CREATE OR REPLACE FUNCTION total()
RETURNS numeric AS $$
DECLARE
    tmp RECORD; result numeric;
BEGIN
    result := 0.00;
    FOR tmp IN SELECT * FROM foo LOOP
        result := result + tmp.f1;
    END LOOP;
    RETURN result;
END;
$$ LANGUAGE plpgsql;

SELECT total();
totalbalance
-----
83.00
```

● 判断和递归

```
CREATE OR REPLACE FUNCTION factorial (i numeric)
RETURNS numeric AS $$
BEGIN
    IF i = 0 THEN
        RETURN 1;
    ELSIF i = 1 THEN
        RETURN 1;
    ELSE
        RETURN i * factorial(i - 1);
    END IF;
END;
$$ LANGUAGE plpgsql;

SELECT factorial(42::numeric);
factorial
-----
140500611775287989854314260624451156993538422002000
```

Greenplum 中文社区所有

2.3.2 常见的 UDF 语言

UDF

GP支持的过程语言

- SQL
- C
- PL/pythonU
- PL/R
- PL/JavaU 和 PL/Java
- PL/perlU 和 PL/Perl
- PL/Container
 - Trusted Python和R

基于UDF实现的应用

- GPText
- MADLib
- DiskQuota
- PostGIS

Greenplum 中文社区所有

2.4 External Web Table

segmentfault 社区

External Web Table

- GP独有的便利功能
- 可以将外部命令分发到各个Segment上并行执行
- 将外部命令的输出整合成结果集，可以连接和聚集
- 需要管理员权限才能创建

```
CREATE EXTERNAL WEB TABLE clean_old_logs_segment(filename text)
EXECUTE '
    find pg_log -maxdepth 1 -name gpdb*.csv -mtime +7 -type f -delete -print
' FORMAT 'TEXT';

SELECT gp_segment_id, * FROM clean_old_logs_segment;
```

gp_segment_id	filename
0	pg_log/gpdb-2018-12-06_000000.csv
0	pg_log/gpdb-2018-12-07_000000.csv
1	pg_log/gpdb-2018-12-06_000000.csv
1	pg_log/gpdb-2018-12-07_000000.csv
2	pg_log/gpdb-2018-12-06_000000.csv
2	pg_log/gpdb-2018-12-07_000000.csv

Greenplum 中文社区所有

2.5 公用表表达式 CTE

CTE

- SQL 99 标准
- 分为普通CTE和Recursive CTE
- 普通CTE可以帮助简化查询语句，方便阅读书写
- 使用CTE并不会影响查询计划和性能

```
CREATE TABLE demo (
    id INT
    , ctime TIMESTAMP
);

CREATE TABLE demo2 (
    id INT
    , status INT
    , name TEXT
);
```

```
SELECT status, name
FROM demo JOIN demo2
ON demo.id = demo2.id
WHERE name = 'A'
AND status = 2
AND ctime = (
    SELECT max(ctime)
    FROM demo JOIN demo2
    ON demo.id = demo2.id
    WHERE name = 'A'
    AND status = 2
);
```

相似的结构

Greenplum 中文社区所有

CTE

- SQL 99 标准
- 分为普通CTE和Recursive CTE
- 普通CTE可以帮助简化查询语句，方便阅读书写
- 使用CTE并不会影响查询计划和性能

```
CREATE TABLE demo (  
  id INT  
  , ctime TIMESTAMP  
);
```

```
CREATE TABLE demo2 (  
  id INT  
  , status INT  
  , name TEXT  
);
```

```
SELECT status, name  
FROM demo JOIN demo2  
ON demo.id = demo2.id  
WHERE name = 'A'  
  AND status = 2  
  AND ctime = (  
    SELECT max(ctime)  
    FROM demo JOIN demo2  
    ON demo.id = demo2.id  
    WHERE name = 'A'  
    AND status = 2  
  );
```



```
WITH cte_demo (ctime, status, name)  
AS (  
  SELECT ctime, status, name  
  FROM demo JOIN demo2  
  ON demo.id = demo2.id  
  WHERE name = 'A'  
    AND status = 2  
)  
SELECT * FROM cte_demo  
WHERE ctime = (  
  SELECT max(ctime) FROM cte_demo  
);
```

Greenplum 中文社区所有

Recursive CTE

- 举例：分析公司员工的汇报关系



Greenplum 中文社区所有

Recursive CTE

- 怎样通过SQL分析得出如下结果？
求每个员工的直属下属和间接下属总数

员工ID	姓名	下属总数
10000	张三	16
20001	李四	8
20005	王五	6
20008	赵六	3
21001	小A	0

.....

Greenplum 中文社区所有

Recursive CTE

解答:

```
WITH RECURSIVE rccte_demo (id, name, manager_id, cnt)
AS (
  SELECT m.id, m.name, m.manager_id, count(*)
  FROM employee m JOIN employee r
    ON m.id = r.manager_id
  GROUP BY (m.id, m.name, m.manager_id)
  UNION ALL
  SELECT e.id, e.name, e.manager_id, r.cnt
  FROM employee e JOIN rccte_demo r
    ON e.id = r.manager_id
)
SELECT id, name, sum(cnt) rcnt
FROM rccte_demo
GROUP BY (id, name)
ORDER BY rcnt desc, id, name;
```

初始
查询递归
查询

附：范例数据

```
CREATE TABLE employee (
  id      INT
, name    TEXT
, level   TEXT
, manager_id INT
);

INSERT INTO employee VALUES
(10000, 'ZHANG SAN', 'CEO', NULL),
(20001, 'LI SI', 'VP', 10000);
(20005, 'WANG WU', 'VP', 10000);
(20008, 'ZHAO LIU', 'M', 20001);
(21001, 'A', 'E', 20008);
(21002, 'B', 'E', 20008);
(21003, 'C', 'E', 20008);
(20231, 'JIA', 'E', 20001);
(24122, 'YI', 'E', 20001);
(30123, 'BING', 'E', 20001);
(30211, 'DING', 'E', 20001);
(43421, 'EAST', 'E', 20005);
(31353, 'WEST', 'E', 20005);
(23112, 'SOUTH', 'E', 20005);
(28995, 'NORTH', 'E', 20005);
(63421, 'GUO JING', 'E', 28995);
(21004, 'HONG NONG', 'E', 21001);
```

Greenplum 中文社区所有

2.6 ROLLUP,CUBE,and GROUPING SETS

segmentfault 课堂

Rollup, Cube 和 Grouping Sets

- SQL 99 标准
- 用于对一组事实的多维度透视，常见于BI工具
- 分析型数据库



图片来自于网络

Greenplum 中文社区所有

2.7 窗口函数

segmentfault 课堂

窗口函数

- SQL 2003 标准

```
CREATE TABLE student_score (
    student TEXT
    , class TEXT
    , score INT
);

INSERT INTO student_score VALUES
('A', 'MATH', 98),
('B', 'MATH', 60),
('C', 'MATH', 100),
('D', 'MATH', 95),
('E', 'MATH', 70),
('A', 'ENGLISH', 90),
('B', 'ENGLISH', 99),
('X', 'ENGLISH', 80),
('Y', 'ENGLISH', 59);
```

举例：求每个科目的前三名

```
SELECT * FROM (
    SELECT
        *,
        rank() OVER (
            PARTITION BY class
            ORDER BY score DESC) rn
    FROM student_score
) dt
WHERE rn <= 3;
```

student	class	score	rn
B	ENGLISH	99	1
A	ENGLISH	90	2
X	ENGLISH	80	3
C	MATH	100	1
A	MATH	98	2
D	MATH	95	3

Greenplum 中文社区所有

2.8 GP 扩展介绍

详细的请查看：

GP扩展框架

- PG有着丰富的扩展框架生态
 - <https://github.com/postgres/postgres/tree/master/contrib>
- 基于扩展框架产品
 - GPCC Metrics Collector 指标收集器
 - GPCC Workload Manager 工作负载管理
 - DiskQuota 磁盘配额
 - PostGIS 地理信息
 - pgcrypto 数据加密
 - pgaudit 数据审计

Greenplum 中文社区所有

扩展框架举例

GP扩展组件的构成

- Makefile
- control文件
- sql文件
- C代码文件

my_extension.control

```
comment = 'Sample extension'
default_version = '1.0'
module_pathname = '$libdir/my_extension'
relocatable = true
```

my_extension--1.0.sql

```
CREATE FUNCTION my_extension_demo()
RETURNS TEXT STRICT
AS 'MODULE_PATHNAME'
LANGUAGE C;
```

Makefile

```
MODULES = my_extension

EXTENSION = my_extension
DATA = my_extension--1.0.sql

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

my_extension.c

```
#include "postgres.h"
#include "utils/builtins.h"

PG_MODULE_MAGIC;

PG_FUNCTION_INFO_V1(my_extension_demo);

Datum
my_extension_demo(PG_FUNCTION_ARGS)
{
    PG_RETURN_TEXT_P(
        cstring_to_text("Hello Greenplum!")
    );
}
```

Greenplum 中文社区所有

扩展框架举例

● make

```
# I6X_STABLE + origin 7:3 xl + make
gcc -Wall -Wmissing-prototypes -Wpointer-arith -Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing -fwrapv -Wno-address -Wno-unused-command-line-argument -g -ggdb -O2 -std-gnu99 -Werror-uninitialized -Werror-implicit-function-declaration -I. -I./ -I/Users/haowang/usr/local/greenplum-db-6X/include/postgresql/server -I/Users/haowang/usr/local/greenplum-db-6X/include/postgresql/internal -c -o my_extension.o my_extension.c -MD -MP -MF deps/my_extension.Po
gcc -Wall -Wmissing-prototypes -Wpointer-arith -Wendif-labels -Wmissing-format-attribute -Wformat-security -fno-strict-aliasing -fwrapv -Wno-address -Wno-unused-command-line-argument -g -ggdb -O2 -std-gnu99 -Werror-uninitialized -Werror-implicit-function-declaration -I/Users/haowang/usr/local/greenplum-db-6X/lib -Wl,-dead_strip_dylibs -bundle -bundle_loader /Users/haowang/usr/local/greenplum-db-6X/bin/postgres -o my_extension.so my_extension.o
```

● make install

```
2019-08-12 20:22:57 @ Haas-MacBook-Pro in ~/workspace/gpdb/contrib/my_extension
# I6X_STABLE + origin 7:3 xl + make install
/bin/sh /Users/haowang/usr/local/greenplum-db-6X/lib/postgresql/pgxs/src/makefiles/../../config/install-sh -c -d '/Users/haowang/usr/local/greenplum-db-6X/share/postgresql/extension'
/bin/sh /Users/haowang/usr/local/greenplum-db-6X/lib/postgresql/pgxs/src/makefiles/../../config/install-sh -c -d '/Users/haowang/usr/local/greenplum-db-6X/share/postgresql/extension'
/bin/sh /Users/haowang/usr/local/greenplum-db-6X/lib/postgresql/pgxs/src/makefiles/../../config/install-sh -c -d '/Users/haowang/usr/local/greenplum-db-6X/lib/postgresql'
/usr/bin/install -c -m 644 my_extension.control '/Users/haowang/usr/local/greenplum-db-6X/share/postgresql/extension/'
/usr/bin/install -c -m 644 my_extension-1.0.sql '/Users/haowang/usr/local/greenplum-db-6X/share/postgresql/extension/'
/usr/bin/install -c -m 755 my_extension.so '/Users/haowang/usr/local/greenplum-db-6X/lib/postgresql/'
```

```
2019-08-12 20:26:38 @ Haas-MacBook-Pro in ~/workspace/gpdb/contrib/my_extension
# I6X_STABLE + origin 7:3 xl + psql -c "CREATE EXTENSION my_extension;"
CREATE EXTENSION

2019-08-12 20:30:22 @ Haas-MacBook-Pro in ~/workspace/gpdb/contrib/my_extension
# I6X_STABLE + origin 7:3 xl + psql -c "SELECT * FROM my_extension_demo();"
my_extension_demo
-----
Hello Greenplum
(1 row)
```

Greenplum 中文社区所有