PostgreSQL：使用clickhousedb\_fdw访问最快列式DBMS之一的ClickHouse

出自，[PostgreSQL: Access ClickHouse, One of the Fastest Column DBMSs, With clickhousedb\_fdw](https://www.percona.com/blog/2019/03/29/postgresql-access-clickhouse-one-of-the-fastest-column-dbmss-with-clickhousedb_fdw/)

翻译， alitrack

校对，略仔

数据库管理系统旨在容纳数据，但偶尔也可能需要与另一个DBMS通信。我们访问的外部服务器可能托管了另外的DBMS。随着这样的异构环境变得越来越普遍，用于这些服务器之间的桥梁被建立了。我们将此称为称为“外部数据包装器”（FDW）。 PostgresSQL在2013年发布的9.3的版本中完成了对SQL/MED的支持。 外部数据封装器是由PostgreSQL服务器加载的共享库。它允许在PostgreSQL中创建外部表，作为另一个数据源的代理。

查询外表时，Postgres将请求传递给关联的外部数据封装器。 FDW创建连接并检索或更新外部数据存储中的数据。由于PostgreSQL规划器也涉及所有这个过程，因此当从数据源检索时，它可能执行某些操作，如聚合或连接数据。我将在本文稍后介绍其中一些内容。

**Clickhouse 数据库**

ClickHouse是一个开源列式数据库管理系统，声称比传统方法快100-1,000倍，能够在不到一秒的时间内处理超过10亿行数据。

**clickhousedb\_fdw**

clickhousedb\_fdw是一个来自Percona的开源项目， 它采用Apache许可。 GitHub项目存储库的链接如下，

<https://github.com/Percona-Lab/clickhousedb_fdw>

它是ClickHouse的FDW，允许你通过PostgreSql v11数据库服务器在ClickHouse数据库中查询或者插入数据。

FDW支持聚合下推和Join下推等高级功能。 通过将远程服务器的资源用于这些资源密集型操作，这些显着提高了性能。

如果您想关注这篇文章并尝试Postgres和ClickHouse之间的FDW，您可以下载并设置[ClickHouse的[ontime数据集]（<https://clickhouse.yandex/docs/en/getting_started/example_datasets/ontime/>）。 按照说明操作后，测试即可获得所需数据。 ClickHouse客户端是ClickHouse数据库的客户端CLI。

**为ClickHouse准备数据**

现在数据在ClickHouse中准备就绪，下一步是设置PostgreSQL。 我们需要创建一个ClickHouse外部服务器，用户映射和外部表。

**安装clickhousedb\_fdw扩展**

虽然可以手动方式安装clickhousedb\_fdw，但clickhousedb\_fdw可以使用PostgreSQL最酷的扩展安装功能。 只需输入SQL命令即可使用扩展，

*CREATE EXTENSION clickhousedb\_fdw;*

创建服务器，

*CREATE SERVER clickhouse\_svr FOREIGN DATA WRAPPER clickhousedb\_fdw OPTIONS(dbname 'test\_database', driver '/use/lib/libclickhouseodbc.so');*

创建用户映射，

*CREATE USER MAPPING FOR CURRENT\_USER SERVER clickhouse\_svr;*

创建外表，

*CREATE FOREIGN TABLE clickhouse\_tbl\_ontime (  "Year" Int,  "Quarter" Int8,  "Month" Int8,  "DayofMonth" Int8,  "DayOfWeek" Int8,  "FlightDate" Date,  "UniqueCarrier" Varchar(7),  "AirlineID" Int,  "Carrier" Varchar(2),  "TailNum" text,  "FlightNum" text,  "OriginAirportID" Int,  "OriginAirportSeqID" Int,  "OriginCityMarketID" Int,  "Origin" Varchar(5),  "OriginCityName" text,  "OriginState" Varchar(2),  "OriginStateFips" text,  "OriginStateName" text,  "OriginWac" Int,  "DestAirportID" Int,  "DestAirportSeqID" Int,  "DestCityMarketID" Int,  "Dest" Varchar(5),  "DestCityName" text,  "DestState" Varchar(2),  "DestStateFips" text,  "DestStateName" text,  "DestWac" Int,  "CRSDepTime" Int,  "DepTime" Int,  "DepDelay" Int,  "DepDelayMinutes" Int,  "DepDel15" Int,  "DepartureDelayGroups" text,  "DepTimeBlk" text,  "TaxiOut" Int,  "WheelsOff" Int,  "WheelsOn" Int,  "TaxiIn" Int,  "CRSArrTime" Int,  "ArrTime" Int,  "ArrDelay" Int,  "ArrDelayMinutes" Int,  "ArrDel15" Int,  "ArrivalDelayGroups" Int,  "ArrTimeBlk" text,  "Cancelled" Int8,  "CancellationCode" Varchar(1),  "Diverted" Int8,  "CRSElapsedTime" Int,  "ActualElapsedTime" Int,  "AirTime" Int,  "Flights" Int,  "Distance" Int,  "DistanceGroup" Int8,  "CarrierDelay" Int,  "WeatherDelay" Int,  "NASDelay" Int,  "SecurityDelay" Int,  "LateAircraftDelay" Int,  "FirstDepTime" text,  "TotalAddGTime" text,  "LongestAddGTime" text,  "DivAirportLandings" text,  "DivReachedDest" text,  "DivActualElapsedTime" text,  "DivArrDelay" text,  "DivDistance" text,  "Div1Airport" text,  "Div1AirportID" Int,  "Div1AirportSeqID" Int,  "Div1WheelsOn" text,  "Div1TotalGTime" text,  "Div1LongestGTime" text,  "Div1WheelsOff" text,  "Div1TailNum" text,  "Div2Airport" text,  "Div2AirportID" Int,  "Div2AirportSeqID" Int,  "Div2WheelsOn" text,  "Div2TotalGTime" text,  "Div2LongestGTime" text,"Div2WheelsOff" text,  "Div2TailNum" text,  "Div3Airport" text,  "Div3AirportID" Int,  "Div3AirportSeqID" Int,  "Div3WheelsOn" text,  "Div3TotalGTime" text,  "Div3LongestGTime" text,  "Div3WheelsOff" text,  "Div3TailNum" text,  "Div4Airport" text,  "Div4AirportID" Int,  "Div4AirportSeqID" Int,  "Div4WheelsOn" text,  "Div4TotalGTime" text,  "Div4LongestGTime" text,  "Div4WheelsOff" text,  "Div4TailNum" text,  "Div5Airport" text,  "Div5AirportID" Int,  "Div5AirportSeqID" Int,  "Div5WheelsOn" text,  "Div5TotalGTime" text,  "Div5LongestGTime" text,  "Div5WheelsOff" text,  "Div5TailNum" text) server clickhouse\_svr options(table\_name 'ontime');*

查询样例，

*postgres=# SELECT a."Year", c1/c2 as Value FROM ( select "Year", count(\*)\*1000 as c1            
          FROM clickhouse\_tbl\_ontime            
          WHERE "DepDelay">10 GROUP BY "Year") a                          
          INNER JOIN (select "Year", count(\*) as c2 from clickhouse\_tbl\_ontime            
          GROUP BY "Year" ) b on a."Year"=b."Year" LIMIT 3;  
Year |   value      
------+------------  
1987 |        199  
1988 | 5202096000  
1989 | 5041199000  
(3 rows)*

**性能特性**

PostgreSQL通过添加下推功能改进了外部数据封装器处理。 由于数据处理在处理链中较早发生，因此下推可显着提高性能。 下推能力包括：

* 算子和函数下推
* 谓词下推
* 聚合下推
* Join下推

**算子和函数下推**

函数和算子发送到Clickhouse而不是在PostgreSQL端计算和过滤。

*postgres=# EXPLAIN VERBOSE SELECT avg("DepDelay") FROM clickhouse\_tbl\_ontime WHERE "DepDelay" <10;   
          Foreign Scan  (cost=1.00..-1.00 rows=1000 width=32) Output: (avg("DepDelay"))    
          Relations: Aggregate on (clickhouse\_tbl\_ontime)    
          Remote SQL: SELECT avg("DepDelay") FROM "default".ontime WHERE (("DepDelay" < 10))(4 rows)*

**谓词下推**

Instead of filtering the data at PostgreSQL, clickhousedb\_fdw send the predicate to Clikhouse Database.

clickhousedb\_fdw不会在PostgreSQL端过滤的数据，而是将谓词发送给Clikhouse数据库。

*postgres=# EXPLAIN VERBOSE SELECT "Year" FROM clickhouse\_tbl\_ontime WHERE "Year"=1989;                                     
            
          Foreign Scan on public.clickhouse\_tbl\_ontime Output: "Year"     
          Remote SQL: SELECT "Year" FROM "default".ontime WHERE (("Year" = 1989)*

**聚合下推**

聚合下推是PostgreSQL FDW的一项新功能。 目前很少有外部数据封装器支持聚合下推 - clickhousedb\_fdw就是其中之一。 规划器决定哪些聚合被下推，哪些聚合不下推。 以下是两种情况的示例，

*postgres=# EXPLAIN VERBOSE SELECT count(\*) FROM clickhouse\_tbl\_ontime;  
        Foreign Scan (cost=1.00..-1.00 rows=1000 width=8)  
        Output: (count(\*)) Relations: Aggregate on (clickhouse\_tbl\_ontime)  
        Remote SQL: SELECT count(\*) FROM "default".ontime*

**Join下推**

同样，这也是PostgreSQL FDW中的一个新功能，我们的clickhousedb\_fdw也支持join 下推。 例子如下，

*postgres=# EXPLAIN VERBOSE SELECT a."Year"  
                          FROM clickhouse\_tbl\_ontime a  
                          LEFT JOIN clickhouse\_tbl\_ontime b ON a."Year" = b."Year";  
      Foreign Scan (cost=1.00..-1.00 rows=1000 width=50);  
      Output: a."Year" Relations: (clickhouse\_tbl\_ontime a) LEFT JOIN (clickhouse\_tbl\_ontime b)  
      Remote SQL: SELECT r1."Year" FROM&nbsp; "default".ontime r1 ALL LEFT JOIN "default".ontime r2 ON (((r1."Year" = r2."Year")))*

**Percona's对PostgreSQL的支持**

作为我们致力于成为开源数据库生态系统的公正支持者的一部分承诺，Percona为PostgreSQL提供支持 - [你可以在这里阅读更多相关内容](https://www.percona.com/services/support/postgresql)。 正如您所看到的，作为我们支持承诺的一部分，我们现在正在开发自己的开源PostgreSQL项目，例如clickhousedb\_fdw。 欢迎订阅博客，第一时间从Percona’s了解了解PostgreSQL和其他开源项目。

作为新clickhousdb\_fdw和其他FDW的作者 - 我会很高兴听到您的使用案例和使用此功能的经验。