

Presto

1. Presto背景

2011年，FaceBook的数据仓库存储在少量大型hadoop/hdfs集群，在这之前，FaceBook的科学家和分析师一直靠hive进行数据分析，但hive使用MR作为底层计算框架，是专为批处理设计的，但是随着数据的不断增多，使用hive进行一个简单的数据查询可能要花费几分钟或者几个小时，显然不能满足查询需求，Facebook也调研了其他比hive更快的工具，但是他们需要在功能有限的条件下做简单操作，以至于无法操作Facebook庞大的数据要求。

2012年开始研究自己的框架--presto，每日可以超过1pb查询，而且速度比较快，faceBook声称Presto的性能比hive要好上10倍或者100倍，presto和hive都是facebook开发的。2013年Facebook正式宣布开源Presto。

2. Presto简介

Presto官方网站：<https://prestodb.io/>

**Distributed SQL
Query Engine
for Big Data**

GET STARTED **DOWNLOAD**

```
$ presto
presto:default> describe nation;
  Column | Type | Null | Partition Key |
-----+-----+-----+-----+
n_nationkey | bigint | true | false |
n_name | varchar | true | false |
n_regionkey | bigint | true | false |
n_comment | varchar | true | false |
(4 rows)

Query 20131105_005529_00080_ee7y3, FINISHED, 2 nodes
Splits: 2 total, 2 done (100.00%)
0:00 [8 rows, 446B] [23 rows/s, 1.29KB/s]

presto:default> █
```

大数据分布式查询引擎

Presto是一个开源的分布式SQL查询引擎，适用于交互式查询，数据量支持GB到PB字节。

Presto的设计和编写完全是为了解决Facebook这样规模的商业数据仓库交互式分析和处理速度的问题。

Presto支持在线数据查询，包括Hive、kafka、Cassandra、关系数据库以及专门数据存储，一条Presto查询可以将多个数据源进行合并，可以跨越整个组织进行分析。

Presto以分析师的需求作为目标，他们期望相应速度小于1秒到几分钟，Presto终结了数据分析的两难选择，要么使用速度快的昂贵的商业方案，要么使用消耗大量硬件的慢速的“免费”方案。

3. 性能对比

这里主要对比目前流行的几个大数据查询引擎：Hive、SparkSQL、Presto、Impala、HAWQ、ClickHouse、Greenplum。

3.1 Hive

3.1.1 介绍

Hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供完整的sql查询功能，可以将SQL语句转换为MapReduce任务进行运行。其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

Hive是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。

Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 Mapper 和 Reducer 来处理内建的 Mapper 和 Reducer 无法完成的复杂的分析工作。

3.1.2 性能

Hive相对于其他查询引擎来说性能一般，主要的优势体现在系统负载低、稳定性高、数据格式支持面广、社区活跃度高，可以为其他多款查询引擎提供底层元数据，SparkSql、Presto、Impala、HAWQ等都支持基于Hive的查询。

成本低、稳定性好，生态兼容性好，因此Hive在企业中应用的较多。

3.2 SparkSQL

3.2.1 介绍

SparkSQL是Hadoop中另一个著名的SQL引擎，它以Spark作为底层计算框架，Spark使用RDD作为分布式程序的工作集合，它提供一种分布式共享内存的受限形式。在分布式共享内存系统中，应用可以向全局地址空间的任意位置进行读写操作，而RDD是只读的，对其只能进行创建、转化和求值等操作。这种内存操作大大提高了计算速度。

SparkSQL作为Spark生态的一员继续发展，而不再受限于Hive，只是兼容Hive。可以利用hive作为数据源，Spark作为计算引擎，通过SQL解析引擎，实现基于Hive数据源，Spark作为计算引擎的方案。

3.2.2 性能

SparkSQL的性能相对其他的组件要差一些，多表查询性能都不突出。

3.3 Impala

3.3.1 介绍

Impala是Cloudera在受到Google的Dremel启发下开发的实时交互SQL大数据查询工具，它拥有和Hadoop一样的可扩展性、它提供了类SQL（类Hsql）语法，在多用户场景下也能拥有较高的响应速度和吞吐量。它是由Java和C++实现的，Java提供的查询交互的接口和实现，C++实现了查询引擎部分，除此之外，Impala还能够共享Hive Metastore，甚至可以直接使用Hive的JDBC jar和beeline等直接对Impala进行查询、支持丰富的数据存储格式（Parquet、Avro等）。

此外，Impala 没有再使用缓慢的 Hive+MapReduce 批处理，而是通过使用与商用并行关系数据库中类似的分布式查询引擎，可以直接从 HDFS 或 HBase 中用 SELECT、JOIN 和统计函数查询数据，从而大大降低了延迟。

3.3.2 性能

Impala官方宣传其计算速度是一大优点，在实际测试中它的多表查询性能和presto差不多，但是单表查询方面却不如presto好。而且Impala有很多不支持的地方，例如：不支持update、delete操作，不支持grouping sets语法，不支持Date数据类型，不支持ORC文件格式等等，所以impala一般采用Parquet格式进行查询，而且Impala在查询时占用的内存很大。

3.4 HAWQ

3.4.1 介绍

HAWQ 是一个 Hadoop 上的 SQL 引擎，是以 Greenplum Database 为代码基础逐渐发展起来的。HAWQ 采用 MPP 架构，改进了针对 Hadoop 的基于成本的查询优化器。除了能高效处理本身的内部数据，还可通过 PXF 访问 HDFS、Hive、HBase、JSON 等外部数据源。HAWQ 全面兼容 SQL 标准，能编写 SQL UDF，还可用 SQL 完成简单的数据挖掘和机器学习。无论是功能特性，还是性能表现，HAWQ 都比较适用于构建 Hadoop 分析型数据仓库应用。

3.4.2 性能

HAWQ 吸收了先进的基于成本的 SQL 查询优化器，自动生成执行计划，可优化使用 Hadoop 集群资源。HAWQ 采用 Dynamic Pipelining 技术解决这一关键问题。Dynamic Pipelining 是一种并行数据流框架，利用线性可扩展加速 Hadoop 查询，数据直接存储在 HDFS 上，并且其 SQL 查询优化器已经为基于 HDFS 的文件系统性能特征进行过细致的优化。

但是 HAWQ 在多表查询时比 Presto、Impala 差一些；而且不适合单表的复杂聚合操作，单表测试性能方面要比其余四种组件差很多，HAWQ 环境搭建也会遇到诸多问题。

3.5 ClickHouse

3.5.1 介绍

ClickHouse 由俄罗斯 Yandex 公司开发。专为在线数据分析而设计。Yandex 是俄罗斯搜索引擎公司。官方提供的文档表名，ClickHouse 日处理记录数“十亿级”。

特性：

- 采用列式存储
- 数据压缩
- 基于磁盘的存储，大部分列式存储数据库为了追求速度，会将数据直接写入内存，按时内存的空间往往很小
- CPU 利用率高，在计算时会使用机器上的所有 CPU 资源
- 支持分片，并且同一个计算任务会在不同分片上并行执行，计算完成后会将结果汇总
- 支持 SQL，SQL 几乎成了大数据的标准工具，使用门槛较低

- 支持联表查询
- 支持实时更新
- 自动多副本同步
- 支持索引
- 分布式存储查询

3.5.2 性能

ClickHouse 作为目前所有开源MPP计算框架中计算速度最快的，它在做多列的表，同时行数很多的表的查询时，性能是很让人兴奋的，但是在做多表的Join时，它的性能是不如单宽表查询的。

性能测试结果表明ClickHouse在单表查询方面表现出很大的性能优势，但是在多表查询中性能却比较差，不如Presto和Impala、HAWQ的效果好。

3.6 Greenplum

3.6.1 介绍

Greenplum是一个开源的大规模并行数据分析引擎。借助MPP架构，在大型数据集上执行复杂SQL分析的速度比很多解决方案都要快。

特性：

- GPDB完全支持ANSI SQL 2008标准和SQL OLAP 2003 扩展。
- 从应用编程接口上讲，它支持ODBC和JDBC。
- 完善的标准支持使得系统开发、维护和管理都大为方便。
- 支持分布式事务，支持ACID。
- 保证数据的强一致性。
- 做为分布式数据库，拥有良好的线性扩展能力。
- GPDB有完善的生态系统，可以与很多企业级产品集成，譬如SAS、Cognos、Informatic、Tableau等。
- 也可以很多种开源软件集成，譬如Pentaho、Talend 等。

3.6.2 性能

Greenplum作为关系型数据库产品，它的特点主要就是查询速度快，数据装载速度快，批量

DML处理快。而且性能可以随着硬件的添加，呈线性增加，拥有非常良好的可扩展性。因此，它主要适用于面向分析的应用。比如构建企业级ODS/EDW，或者数据集市等，Greenplum都是不错的选择。

整体性能上Greenplum的表现比较中庸，单表查询不如clickhouse，多表查询不如impala，整体性能不如presto。

3.7 Presto

3.7.1 介绍

Presto是一个分布式SQL查询引擎，它被设计为用来专门进行高速、实时的数据分析。它支持标准的ANSI SQL，包括复杂查询、聚合（Aggregation）、连接（Join）和窗口函数（Window Functions）。作为Hive和Pig（Hive和Pig都是通过MapReduce的管道流来完成HDFS数据的查询）的替代者，Presto本身并不存储数据，但是可以接入多种数据源，并且支持跨数据源的级联查询。Presto是一个OLAP的工具，擅长对海量数据进行复杂的分析；但是对于OLTP场景，并不是Presto所擅长，所以不要把Presto当做数据库来使用。

3.7.2 性能

Presto综合性能比起来要比其余组件好一些，无论是查询性能还是支持的数据源和数据格式方面都要突出一些，在单表查询时性能靠前，多表查询方面性能也很突出。

由于Presto是完全基于内存的并行计算，所以Presto在查询时占用的内存也不少，但是要比Impala少一些，比如多表Join时需要很大的内存，Impala占用的内存比Presto要多。

3.8 总结

3.8.1 多表查询

Presto、Impala以及HAWQ在多表查询方面更有优势。

虽说Presto和Impala在多表查询方面的性能差别不大，但是Impala的功能有一些局限性，Impala不支持的功能是没有办法参与性能对比测试的，例如：不支持update、delete操作，不支持grouping sets语法，不支持Date数据类型，不支持ORC文件格式等等，而Presto则基本没有这些局限问题。

3.8.2 单大表聚合

在单表测试方面ClickHouse性能最好，其次是Presto，相比于HAWQ和impala以及SparkSQL在单大表聚合操作方面的表现也相对优秀。

3.8.3 综合对比表格

	hive	impala	presto	sparksql	hawq	clickhouse	greenplum
查询速度快（多表关联）	1	5	4	3	4	3	3
查询速度快（单表查询）	1	3	4	3	3	5	3
系统负载低	4	2	2	2	2	2	2
连接数据源丰富程度	1	3	5	3	3	1	1
支持的数据格式	5	4	5	5	5	3	3
标准sql支持程度	4	4	4	4	5	3	5
系统易用性	5	5	5	4	3	5	5
社区活跃度	5	4	5	5	3	2	4
自定义函数开发周期快	5	4	5	4	4	1	4

3.8.4 使用场景

3.8.4.1 Presto

多数据源时，可以使用presto进行统一查询。

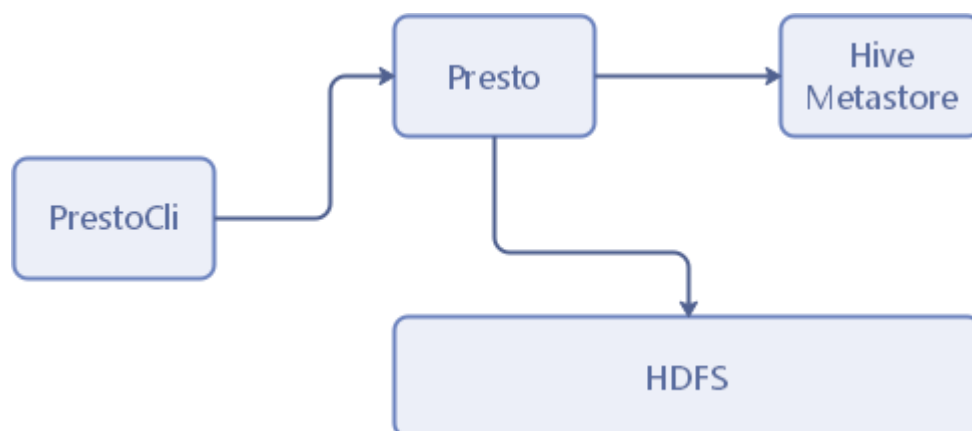
快速查询时，presto查询性能高，但是需要的硬件资源也更昂贵。适合在单次扫描级别GB、TB级别的数据。

多张大表的关联查询时不应该使用presto，presto也不应作为etl工具，因此，在数仓的前两层很少使用presto。

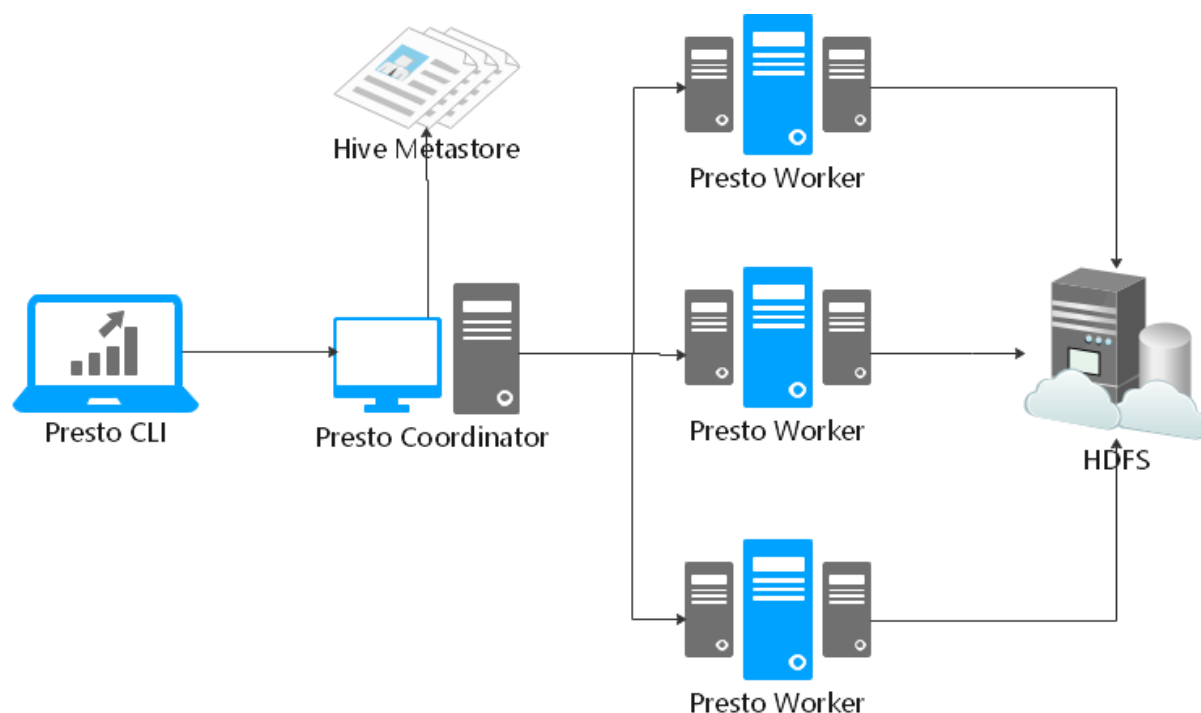
3.8.4.2 Hive

海量数据的场景下，一是需要大量的硬件资源，二是海量的数据极可能造成内存溢出等各种异常。此时推荐使用Hive：成本低、稳定性好，且生态兼容性好。

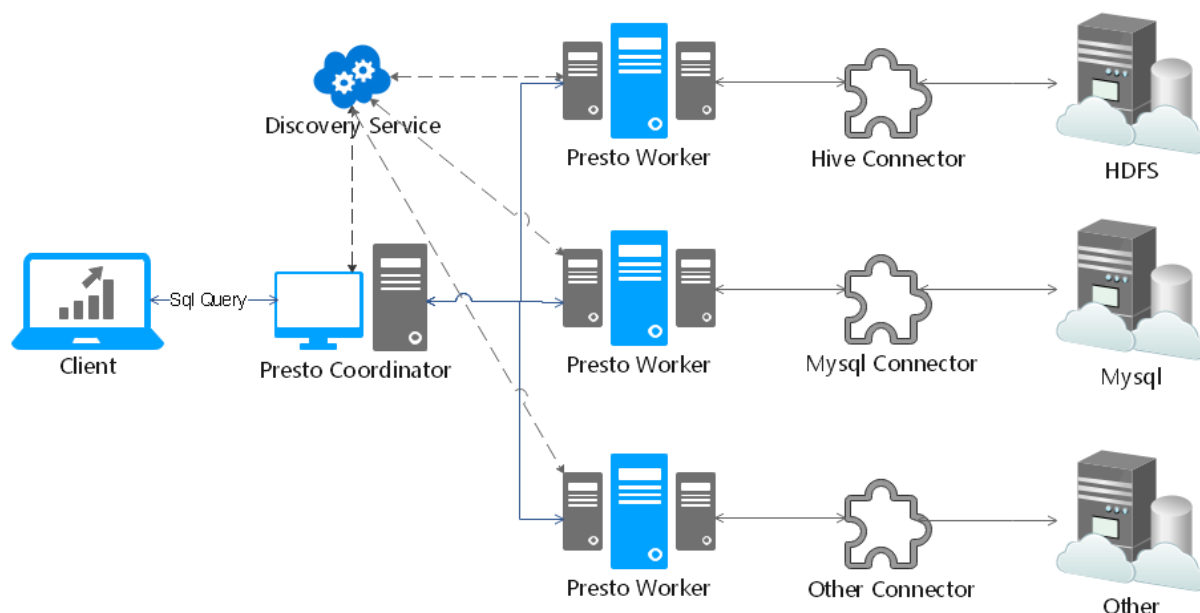
4. Presto架构



Presto是一个运行在多台服务器上的分布式系统。完整安装包括一个coordinator和多个worker。由客户端提交查询，从Presto命令行CLI提交到coordinator。coordinator进行解析，分析并执行查询计划，然后分发处理队列到worker。



Presto查询引擎是一个M-S的架构，由一个coordinator节点，一个Discovery Server节点，多个Worker节点组成，Discovery Server通常内嵌在Coordinator节点中。Coordinator负责SQL的解析，生成执行计划，分发给Worker节点进行执行，Worker节点负责实时查询执行任务。Worker节点启动后向discovery Server服务注册，Coordinator从discovery server获取可以工作的Worker节点。如果配置了hive connector，需要配置hive MetaSote服务为Presto提供元信息，worker节点和HDFS进行交互数据。



5. 相关术语

5.1 Connector 连接器

Presto通过Connector连接器来适应数据源，例如Hive或关系数据库。功能类似于数据库的驱动程序。允许Presto使用标准API与资源进行交互。

Presto包含几个内置连接器：JMX连接器，可访问内置系统表的System连接器，Hive连接器和旨在提供TPC-H基准数据的TPCH连接器。许多第三方开发人员都贡献了连接器，因此Presto可以访问各种数据源中的数据，比如：ES、Kafka、MongoDB、Redis、Postgre、Druid、Cassandra等。

每个Catalog都与一个特定的连接器关联。如果检查Catalog配置文件，将会看到每个都包含一个强制性属性connector.name，Catalog Manager使用此属性指定Catalog的连接器。可能有多个Catalog使用同一连接器来访问相似数据库的两个不同实例。比如，有两个Hive群集，则可以在单个Presto群集中配置两个都使用Hive连接器的Catalog，从而允许从两个不同的Hive集群中查询数据（可以在同一SQL中查询）。

5.2 Catalog 连接目录

Presto Catalog是数据源schema的上一级，并通过连接器访问数据源。例如，可以配置Hive Catalog以通过Hive Connector连接器提供对Hive信息的访问。

在Presto中使用表时，标准表名始终是被支持的。例如，hive.test_data.test的标准表名将引用hive catalog中test_data schema中的test table。

Catalog需要在Presto的配置文件中配置。

5.3 Schema

Schema是组织表的一种方式。Catalog和Schema共同定义了一组可以查询的表。当使用Presto访问Hive或关系数据库（例如MySQL）时，Schema会转换为目标数据库中的对应Schema。

5.4 Table

Table表是一组无序的行，它们被组织成具有类型的命名列。与关系数据库中的含义相同。

6. Presto安装

6.1 Presto-Server安装

6.1.1 环境要求

linux或者MacOS

Java8 64位

Python 2.4+

6.1.2 Java8安装

```
yum install java-1.8.0-openjdk* -y
```

安装完成后，查看jdk版本：

```
java -version
```

6.1.3 下载解压

1、下载安装包

<https://prestodb.io/download.html>

在资料目录中已经下载好。

2、上传presto-server-0.245.1.tar到 hadoop01 的/export/server目录

3、解压

```
tar -xzf presto-server-0.245.1.tar.gz
mv presto-server-0.245.1 presto
```

4、查/看目录结构

```
cd /export/server/presto
```

bin---可执行文件

lib---对应的jar包

plugin---第三方库插件

```
drwxr-xr-x.  3 root root   80 Nov 21  2018 bin
drwxr-xr-x.  2 root root 8192 Nov 21  2018 lib
-rw-r--r--.  1 root root 191539 Nov 21  2018 NOTICE
drwxr-xr-x. 29 root root  4096 Nov 21  2018 plugin
-rw-r--r--.  1 root root   119 Nov 21  2018 README.txt
```

5、Presto需要一个用于存储日志等的data目录。建议在安装目录之外创建一个data目录，以便在升级Presto时可以保留此目录。

6.1.4 配置

在安装目录中创建一个etc目录，此目录下将会包含以下配置文件：

node.properties: 每个节点的环境配置

jvm.config: JVM的命令行选项

config.properties: Presto Server的配置项

catalog/hive.properties: 数据源连接器的配置，此课程将使用hive数据源

6.1.4.1 节点环境配置

etc/node.properties，每个节点的特定配置。

一个节点指的是服务器上Presto的单个已安装实例。

```
node.environment=production
node.id=f7c4bf3c-dbb4-4807-baae-9b7e41807bc8
node.data-dir=/export/server/data
```

node.environment：环境的名称。群集中的所有Presto节点必须具有相同的环境名称。

node.id：此Presto安装的唯一标识符。这对于每个节点都必须是唯一的。在重新启动或升级Presto时，此标识符应保持一致。如果在一台计算机上运行多个Presto安装（即同一台计算机上有多个节点），则每个安装必须具有唯一的标识符。

node.data-dir：数据目录的位置（文件系统路径）。Presto将在此处存储日志和其他数据。

6.1.4.2 JVM虚拟机配置

etc/jvm.config，包含用于启动Java虚拟机的命令行选项列表。文件的格式是选项列表，每行一个。不能使用空格或其他特殊字符。

```
-server
-Xmx5G
-XX:+UseG1GC
-XX:G1HeapRegionSize=32M
-XX:+UseGCOverheadLimit
-XX:+ExplicitGCInvokesConcurrent
-XX:+HeapDumpOnOutOfMemoryError
-XX:+ExitOnOutOfMemoryError
```

由于OutOfMemoryError将会导致JVM处于不一致状态，所以遇到这种错误的时候我们一般的处理措施就是记录下dump heap中的信息（用于debugging），然后强制终止进程。

6.1.4.3 Presto服务配置

etc/config.properties，包含Presto服务器的配置。

Presto服务氛围三种角色：coordinator、worker、coordinator&worker。每个Presto服务都可

以充当coordinator和worker，但是独立出一台服务器专用于coordinator协调工作将在较大的群集上提供最佳性能。

6.1.4.3.1 coordinator配置

```
coordinator=true
node-scheduler.include-coordinator=false
http-server.http.port=8090
query.max-memory=4GB
query.max-memory-per-node=1GB
query.max-total-memory-per-node=2GB
discovery-server.enabled=true
discovery.uri=http://example.net:8090
```

6.1.4.3.2 worker配置

```
coordinator=false
http-server.http.port=8090
query.max-memory=4GB
query.max-memory-per-node=1GB
query.max-total-memory-per-node=2GB
discovery.uri=http://example.net:8090
```

6.1.4.3.3 coordinator&worker配置

```
coordinator=true
node-scheduler.include-coordinator=true
http-server.http.port=8090
query.max-memory=4GB
```

```
query.max-memory-per-node=1GB
query.max-total-memory-per-node=2GB
discovery-server.enabled=true
discovery.uri=http://192.168.88.80:8090
```

单机版使用此配置进行测试。

6.1.4.3.4 配置项含义

coordinator：允许此Presto实例充当coordinator协调器角色（接受来自客户端的查询并管理查询执行）。

node-scheduler.include-coordinator：允许此Presto实例充当coordinator&worker角色。对于较大的群集，coordinator上的worker工作可能会影响查询性能，因为两者互相争抢计算机的资源会导致调度的关键任务受到影响。

http-server.http.port：指定HTTP服务器的端口。Presto使用HTTP进行内部和外部所有通信。

query.max-memory：单个query操作可以使用的最大集群内存量。

query.max-memory-per-node：单个query操作在单个节点上用户内存能用的最大值。

query.max-total-memory-per-node：单个query操作可在单个节点上使用的最大用户内存量和系统内存量，其中系统内存是读取器、写入器和网络缓冲区等在执行期间使用的内存。

discovery-server.enabled：Presto使用发现服务Discovery service来查找群集中的所有节点。每个Presto实例在启动时都会向Discovery服务注册。为了简化部署并避免运行其他服务，Presto协调器coordinator可以运行Discovery服务的嵌入式版本。它与Presto共享HTTP服务器，因此使用相同的端口。

discovery.uri：Discovery服务的URI地址。由于启用了Presto coordinator内嵌的Discovery 服务，因此这个uri就是Presto coordinator的uri。修改example.net:8090，根据你的实际环境设置该URI。此URI不得以"/"结尾。

6.1.4.4 日志级别

etc/log.properties

在这个配置文件中允许你根据不同的日志结构设置不同的日志级别。

Loggers通过名字中的"."来表示层级和集成关系(像java里面的包)。

```
com.facebook.presto=INFO
```

这会将com.facebook.presto.server和com.facebook.presto.hive的日志级别都设置为INFO。

共有四个级别：DEBUG，INFO，WARN和ERROR。

6.1.4.5 连接器配置

Presto通过catalogs中的连接器connectors访问数据。connector提供了对应catalog中的所有schema和table。比如，如果在catalog中配置了Hive connector，并且此Hive的‘web’数据库中有一个‘clicks’表，该表在Presto中就可以通过hive.web.clicks来访问。

通过在etc/catalog目录中创建配置文件来注册connector。比如，通过创建etc/catalog/hive.properties，即可用来注册hive的connector：

```
connector.name=hive-hadoop2
hive.metastore.uri=thrift://192.168.88.80:9083
```

Presto连接器支持以下版本hive：

- 1 . Apache Hive 1.x
- 2 . Apache Hive 2.x
- 3 . Cloudera CHD 4
- 4 . Cloudera CHD 5

Presto的Hive连接器支持的文件类型：

- 1 . ORC
- 2 . Parquet
- 3 . Avro
- 4 . RCFile
- 5 . SequenceFile
- 6 . JSON
- 7 . Text

6.1.4.6 运行Presto

在安装目录的bin/launcher文件，就是启动脚本。Presto可以使用如下命令作为一个后台进程启动：

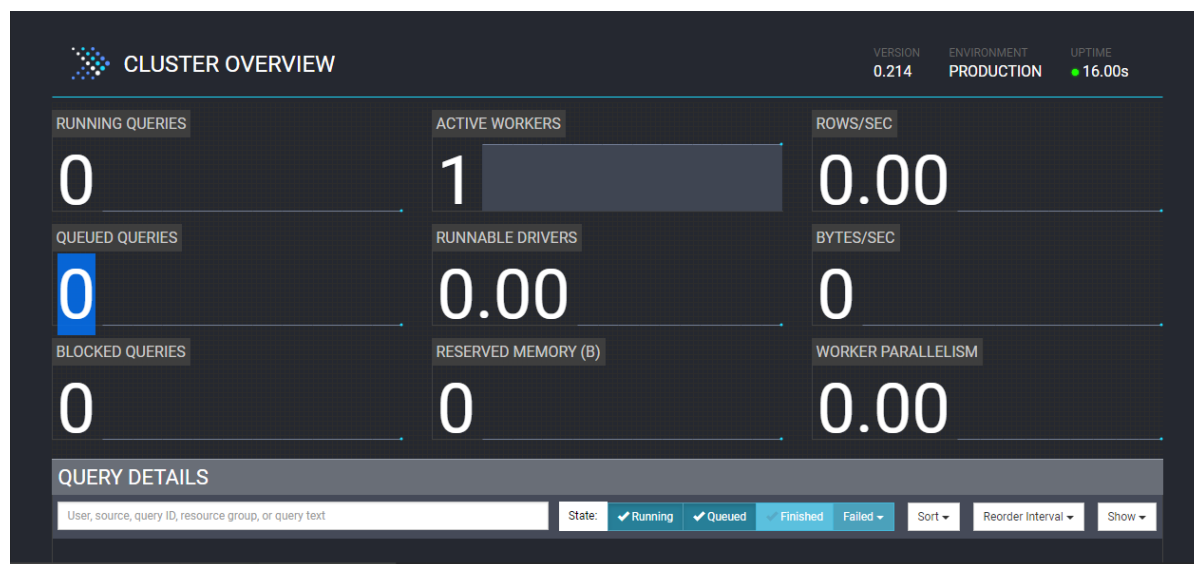

```
bin/launcher start
```

另外，也可以在前台运行，日志和相关输出将会写入stdout/stderr（可以使用类似daemontools的工具捕捉这两个数据流）：

```
bin/launcher run
```

运行bin/launcher --help，Presto将会列出支持的命令和命令行选项。另外可以通过运行时使用--verbose参数，来调试安装是否正确。

访问web：<http://192.168.88.80:8090/ui/>



启动完之后，日志将会写在node.data-dir 配置目录的子目录var/log下，该目录下有如下三个文件：

1. launcher.log：这个日志文件由launcher创建，并且server的stdout和stderr都被重定向到了这个日志文件中。这份日志文件中只会有很少的信息，包括：在server日志系统初始化的时候产生的日志和JVM产生的诊断和测试信息。
2. server.log：这个是Presto使用的主要日志文件。一般情况下，该文件中将会包括server初始化失败时产生的相关信息。这份文件会被自动轮转和压缩。
3. http-request.log：这是HTTP请求的日志文件，包括server收到的每个HTTP请求信息，这份文件会被自动轮转和压缩。

6.2 Presto-CLI安装

Presto CLI提供了基于终端的交互式命令程序，用于运行查询。Presto CLI是一个可执行的JAR文件，这意味着它的行为类似于普通的UNIX可执行文件。

下载presto-cli-0.245.1-executable.jar

(<https://repo1.maven.org/maven2/com/facebook/presto/presto-cli/0.245.1/presto-cli-0.245.1-executable.jar>)，将其重命名为presto，使用chmod +x分配执行权限后，运行：

```
#上传presto-cli-0.245.1-executable.jar到/export/server/presto/bin
mv presto-cli-0.245.1-executable.jar presto
chmod +x presto
./presto --server localhost:8090 --catalog hive --schema default
```

```
[root@hadoop03 bin]# ./presto --server localhost:8090 --catalog hive --schema default
presto:default> show tables;
  Table
-----
 test_orc2
 test_orc3
(2 rows)

Query 20210126_072327_00008_7mxw5, FINISHED, 3 nodes
Splits: 53 total, 53 done (100.00%)
0:01 [2 rows, 52B] [2 rows/s, 61B/s]

presto:default> █
```

6.3 Presto验证

6.3.1 创建数据库(Hive)

```
create database myhive;
```

6.3.2 查看数据库(Hive)

```
show databases;
```

6.3.3 数据准备(Hive)

```
vim employees.txt
```

```
1201 Gopal      45000    Technical manager
1202 Manisha    45000    Proof reader
1203 Masthanvali 40000    Technical writer
1204 Krian      40000    Hr Admin
1205 Kranthi    30000    Op Admin
```

通过Hue将employees.txt文件上传至HDFS的/root/目录下。

6.3.4 创建表(Hive)

```
use myhive;
create table myhive.employee (eud int,name String,salary String,destination String) COMMENT
'Employee table' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' STORED
AS TEXTFILE;
```

6.3.5 加载数据(Hive)

```
load data local inpath '/root/employees.txt' into table employee;
```

6.3.6 查询表(Hive)

```
select * from employee;
```

6.3.7 Presto测试数据(Presto)

```
use myhive;
select * from employee;
```

7. Presto集群搭建

7.1 节点规划

	hadoop01/192.168.88.80	hadoop02/192.168.88.81	hadoop03/192.168.88.82
Coordinator	✓	✗	✗
Worker	✓	✓	✓

7.2 分发配置文件

先在第二台和第三台服务器上创建对应的目录：

```
mkdir -p /export/server
```

在第一台执行复制命令：

```
scp -r presto root@hadoop02:/export/server  
scp -r presto root@hadoop03:/export/server
```

7.3 配置

hadoop01节点作为coordinator节点，其余两个节点为Worker节点。

7.3.1 hadoop01-coordinator配置

7.3.1.1 etc/config.properties

```
coordinator=true  
node-scheduler.include-coordinator=true  
http-server.http.port=8090  
query.max-memory=4GB  
query.max-memory-per-node=1GB  
discovery-server.enabled=true
```

```
discovery.uri=http://192.168.88.80:8090
```

7.3.1.2 etc/jvm.config

```
-server  
-Xmx8G  
-XX:+UseG1GC  
-XX:G1HeapRegionSize=32M  
-XX:+UseGCOverheadLimit  
-XX:+ExplicitGCInvokesConcurrent  
-XX:+HeapDumpOnOutOfMemoryError  
-XX:+ExitOnOutOfMemoryError
```

7.3.1.3 etc/node.properties

```
node.environment=cdhpresto  
node.id=presto-cdh01  
node.data-dir=/export/server/data
```

7.3.1.4 etc/catalog/hive.properties

```
connector.name=hive-hadoop2  
hive.metastore.uri=thrift://192.168.88.80:9083
```

7.3.2 hadoop02/03-worker配置

7.3.2.1 etc/config.properties

```
coordinator=false  
http-server.http.port=8090  
query.max-memory=4GB  
query.max-memory-per-node=1GB  
discovery.uri=http://192.168.88.80:8090
```

7.3.2.2 etc/jvm.config

```
-server  
-Xmx8G  
-XX:+UseG1GC  
-XX:G1HeapRegionSize=32M  
-XX:+UseGCOverheadLimit  
-XX:+ExplicitGCInvokesConcurrent  
-XX:+HeapDumpOnOutOfMemoryError  
-XX:+ExitOnOutOfMemoryError
```

7.3.2.3 etc/node.properties

```
node.environment=cdhpresto  
node.id=presto-cdh02  
node.data-dir=/export/server/data
```

7.3.2.4 etc/catalog/hive.properties

```
connector.name=hive-hadoop2
```

```
hive.metastore.uri=thrift://192.168.88.80:9083
```

7.4 运行

7.4.1 创建快捷启动脚本

在用户根目录创建脚本文件：

```
cd ~  
vim start_presto.sh
```

在文件中写入：

```
/export/server/presto/bin/launcher start
```

分配执行权限：

```
chmod +x start_presto.sh
```

启动三台presto：

```
./start_presto.sh
```

查看是否启动成功：

```
ps -ef|grep presto  
[root@hadoop01 ~]# ps -ef|grep presto  
root      10246      1  26 16:45 ?        00:00:32 java -cp /export/server/presto/lib/* -s  
imit -XX:+ExplicitGCInvokesConcurrent -XX:+HeapDumpOnOutOfMemoryError -XX:+ExitOnOutOfMem  
de.data-dir=/export/server/data -Dnode.id=presto-cdh01 -Dnode.environment=cdhpresto -Dlog  
properties -Dconfig=/export/server/presto/etc/config.properties com.facebook.presto.serve  
root      10576    6277  0 16:47 pts/0    00:00:00 grep --color=auto presto  
[root@hadoop01 ~]#
```

7.4.2 启动Presto客户端

启动三台presto，然后通过CLI分别登录三台server进行测试。

设置presto环境变量：

```
vim /etc/profile
```

在文件末尾写入：

```
export PRESTO_HOME=/export/server/presto  
export PATH=$PATH:$PRESTO_HOME/bin
```

执行生效：

```
source /etc/profile
```


启动presto-cli：

```
presto --server 192.168.88.80:8090 --catalog hive --schema default
```

测试SQL：

```
use myhive;
```

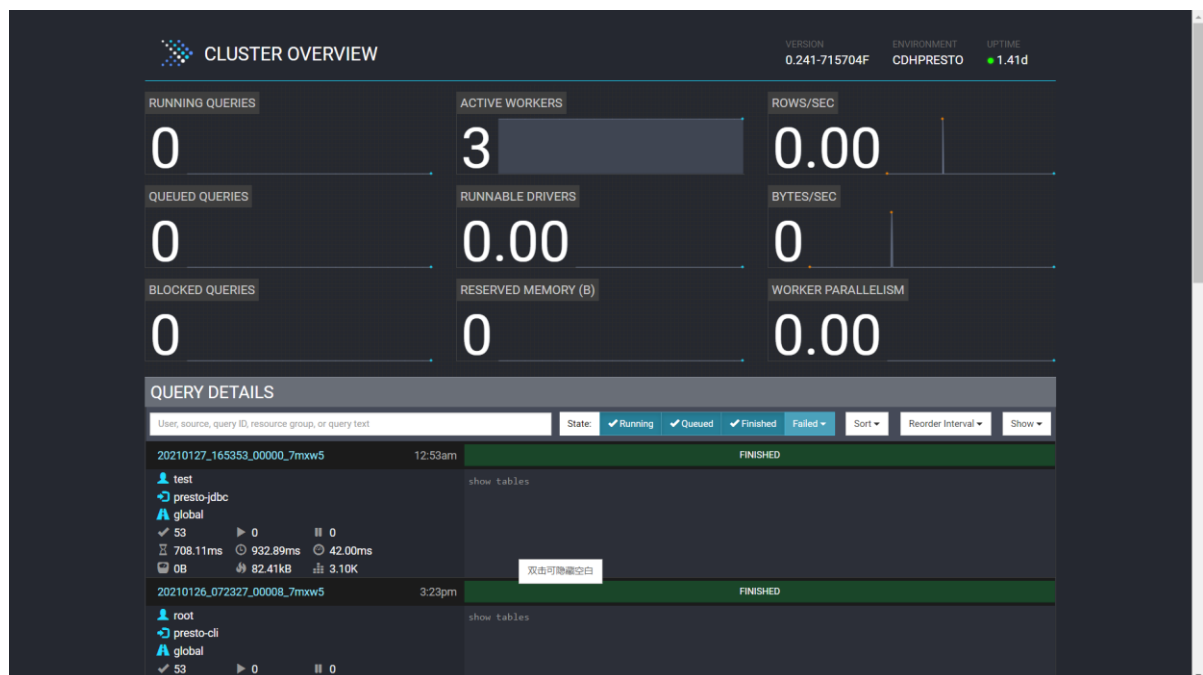
```
select * from employee;
```

```
[root@hadoop01 ~]# presto --server localhost:8090 --catalog hive --schema default
presto:default> use myhive;
USE
presto:myhive> select * from employee;
 eud |      name      | salary | destination
-----+-----+-----+-----
 1201 | Gopal          | 45000  | Technical manager
 1202 | Manisha        | 45000  | Proof reader
 1203 | Masthanvali    | 40000  | Technical writer
 1204 | Krian          | 40000  | Hr Admin
 1205 | Kranthi        | 30000  | Op Admin
(5 rows)

Query 20210408_092813_00006_nd55t, FINISHED, 1 node
Splits: 17 total, 17 done (100.00%)
0:02 [5 rows, 161B] [2 rows/s, 66B/s]

presto:myhive> █
```

7.5 集群管理页面介绍



<http://192.168.88.80:8090/ui/>

主页面显示了正在执行的查询数，正常活动的Worker数，排队的查询数，阻塞的查询数，并

行度等等；以及每个查询的列表区域（包括查询的ID，查询语句，查询状态，用户名，数据源等等）。正在查询的排在最上面，紧接着依次为最近完成的查询，失败的查询等。

查询的状态有以下几种：

QUEUED-查询以及被接受，正等待执行

PLANNING-查询在计划中

STARTING-查询已经开始执行

RUNNING-查询已经运行，至少有一个task开始执行

BLOCKED-查询被阻塞，并且在等待资源（缓存空间，内存，切片）

FINISHING-查询正完成（比如commit for autocommit queries）

FINISHED-查询已经完成（比如数据已输出）

FAILED-查询执行失败

8. Presto JDBC使用

Presto的JDBC使用方法

<https://prestodb.io/resources.html>

Language: Java

Presto JDBC Driver

Project

Presto

Maintained by

Presto Team

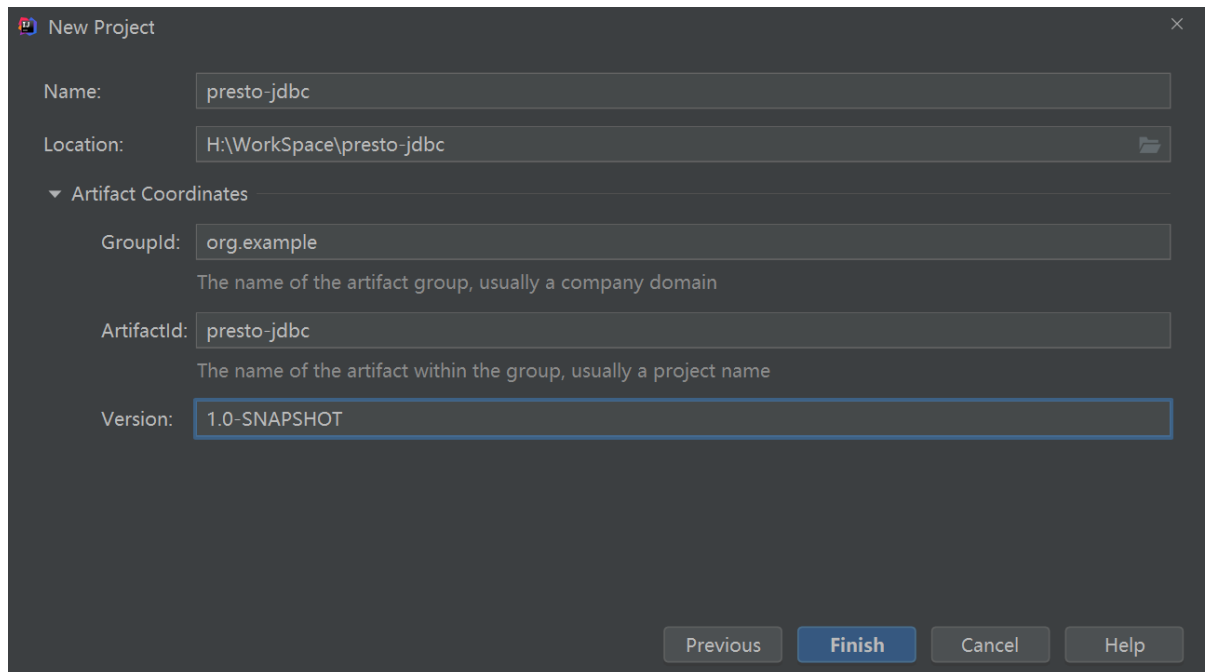
Description

JDBC driver for Presto.

Example

```
String sql = "SELECT * FROM sys.node";
String url = "jdbc:presto://localhost:8080/catalog/schema";
try (Connection connection =
    DriverManager.getConnection(url, "test", null)) {
    try (Statement statement = connection.createStatement()) {
        try (ResultSet rs = statement.executeQuery(sql)) {
            while (rs.next()) {
                System.out.println(rs.getString("node_id"));
            }
        }
    }
}
```

8.1 新建项目



The image shows a 'New Project' dialog box in an IDE. It contains the following fields and values:

- Name:** presto-jdbc
- Location:** H:\WorkSpace\presto-jdbc
- Artifact Coordinates:**
 - GroupId:** org.example (with a tooltip: 'The name of the artifact group, usually a company domain')
 - ArtifactId:** presto-jdbc (with a tooltip: 'The name of the artifact within the group, usually a project name')
 - Version:** 1.0-SNAPSHOT

At the bottom, there are four buttons: 'Previous', 'Finish', 'Cancel', and 'Help'.

8.2 初始化maven依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>presto-jdbc</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>com.facebook.presto</groupId>
      <artifactId>presto-jdbc</artifactId>
```

```
<version>0.245.1</version>

</dependency>

</dependencies>

</project>
```

8.3 Java获取库中表

```
package org.example;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class PrestoJDBC {

    public static void main(String[] args) throws Exception {

        //1.加载驱动
        Class.forName("com.facebook.presto.jdbc.PrestoDriver");

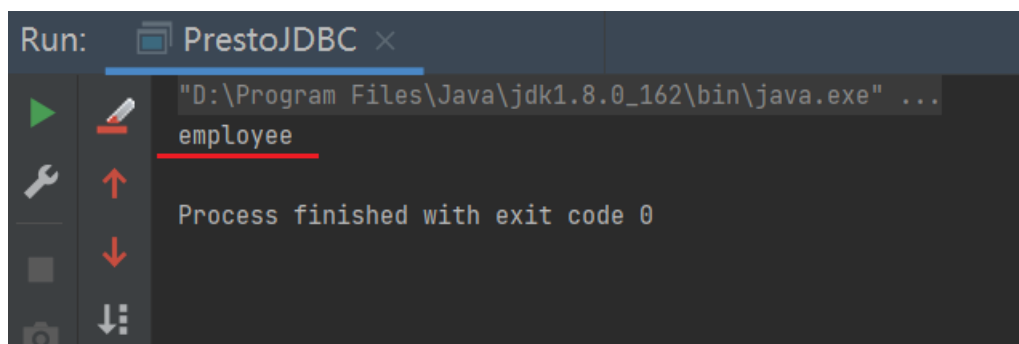
        //2.创建链接
        Connection connection = DriverManager.getConnection("jdbc:presto://192.168.88.80:8090/hive/myhive", "test", null);

        //3.statement
        Statement stmt = connection.createStatement();

        //4.执行SQL
        ResultSet rs = stmt.executeQuery("show tables");
        while (rs.next()) {
            System.out.println(rs.getString(1));
        }

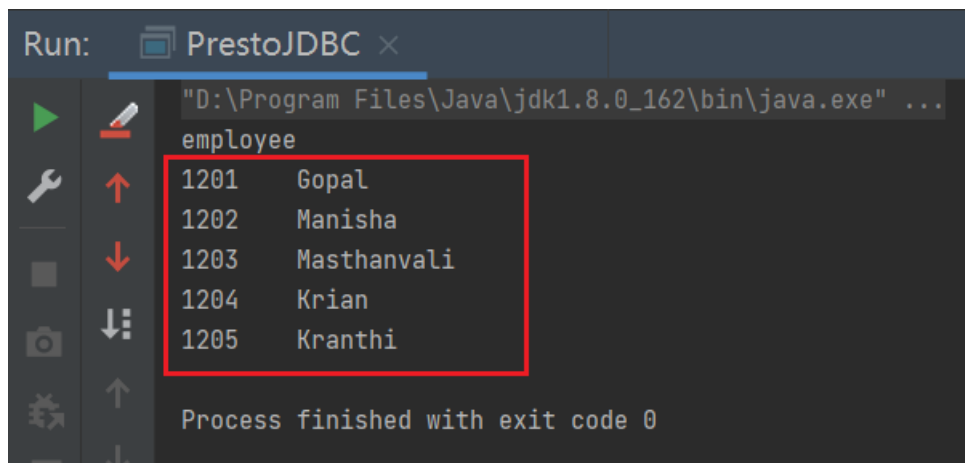
        //5.关闭连接
        rs.close();
        connection.close();
    }
}
```

8.4 运行测试



8.5 Java获取表中数据

```
public static void query() throws SQLException {
    String url = "jdbc:presto://192.168.88.80:8090/hive/myhive";
    Connection connection = null;
    try {
        connection = DriverManager.getConnection(url, "test", null);
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * from employee limit 10");
        while (rs.next()) {
            System.out.println(rs.getString(1)+"\t"+rs.getString(2));
        }
        rs.close();
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    } finally {
        connection.close();
    }
}
```

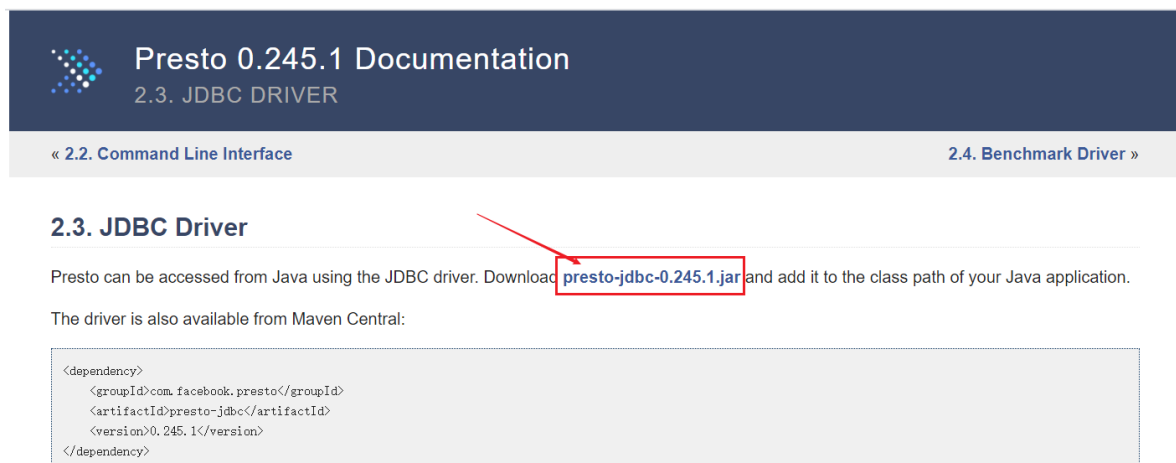


```
Run: PrestoJDBC x
"D:\Program Files\Java\jdk1.8.0_162\bin\java.exe" ...
employee
1201    Gopal
1202    Manisha
1203    Masthanvali
1204    Krian
1205    Kranthi
Process finished with exit code 0
```

9. Presto可视化客户端

9.1 下载驱动

驱动下载：<https://prestodb.io/docs/current/installation/jdbc.html>



Presto 0.245.1 Documentation
2.3. JDBC DRIVER

« 2.2. Command Line Interface 2.4. Benchmark Driver »

2.3. JDBC Driver

Presto can be accessed from Java using the JDBC driver. Download **presto-jdbc-0.245.1.jar** and add it to the class path of your Java application.

The driver is also available from Maven Central:

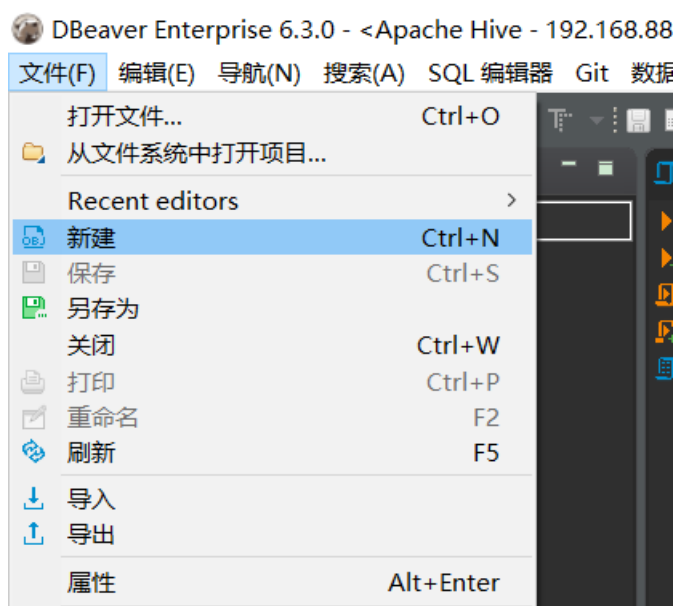
```
<dependency>
  <groupId>com.facebook.presto</groupId>
  <artifactId>presto-jdbc</artifactId>
  <version>0.245.1</version>
</dependency>
```

在【资料/驱动】目录中已下载好驱动包。

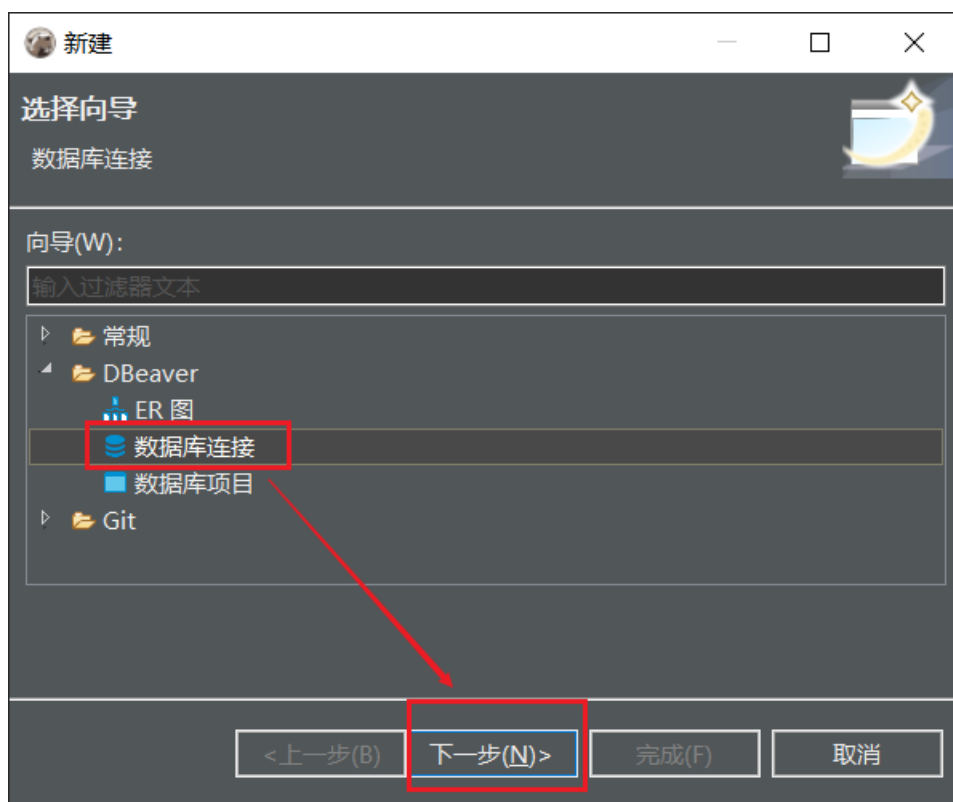
9.2 DBeaver客户端

9.2.1 新建连接

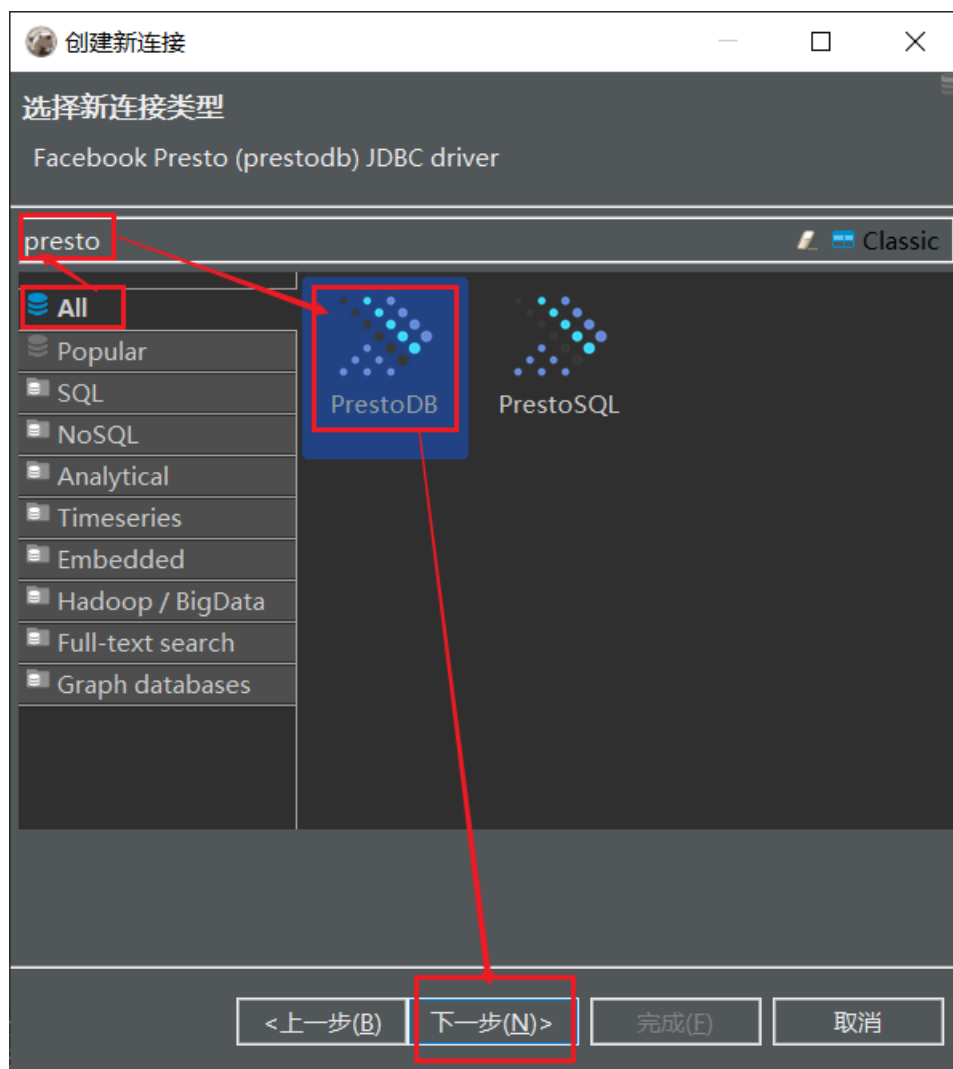
1. 在左上角菜单栏，选择文件->新建：



2. 在弹框中，找到DBeaver下的【数据库连接】，选中，点击【下一步】



3. 选中【ALL】tab，然后在搜索框中键入presto，选中【PrestoDB】，【下一步】：



9.2.2 配置presto驱动

1. 点击【编辑驱动设置】

通用 JDBC 连接设置

PrestoDB 连接设置

常规

驱动属性

SSH

Proxy

JDBC URL: jdbc:presto://localhost:8080

主机: localhost 端口: 8080

数据库/模式:

用户名:

密码: ☐ Save password locally

Advanced settings:

连接详情(名称、类型 ...)

驱动名称: PrestoDB

编辑驱动设置

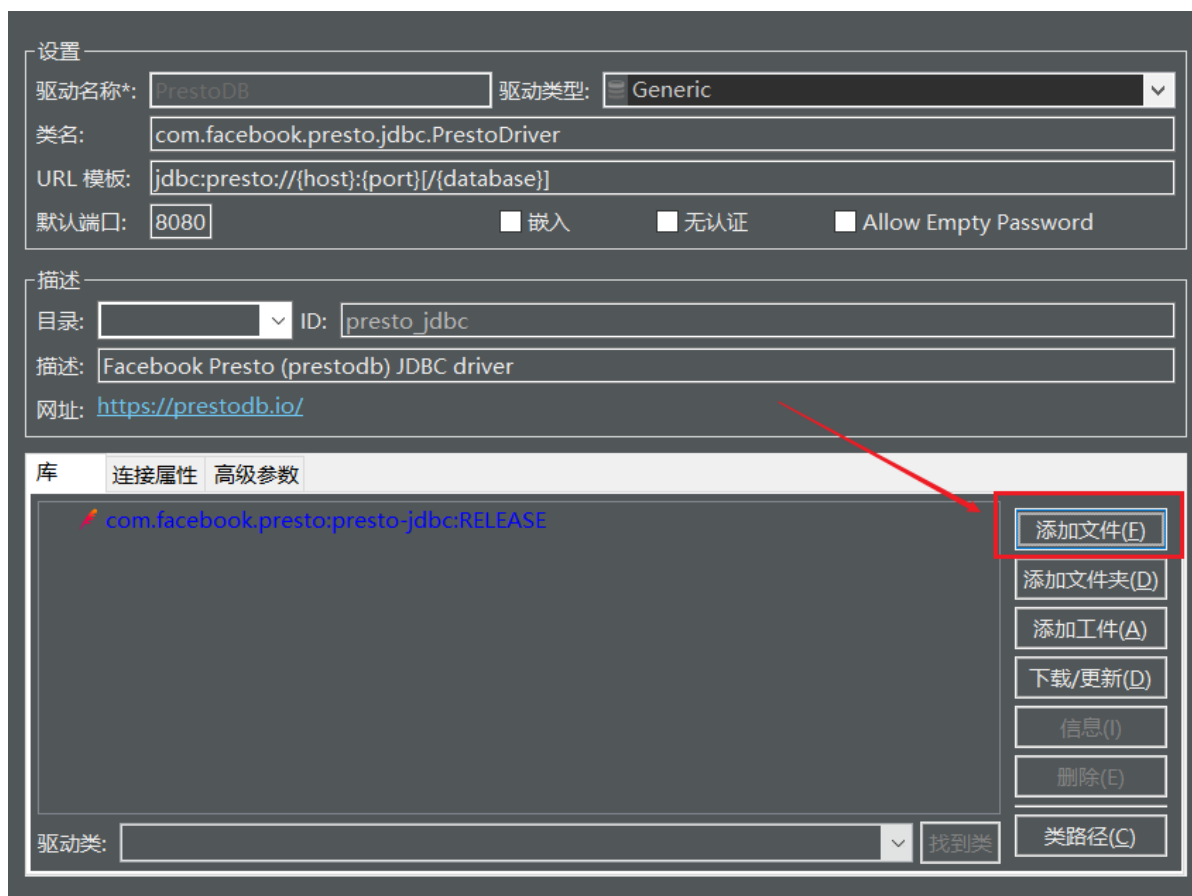
< 上一步(B)

下一步(N) >

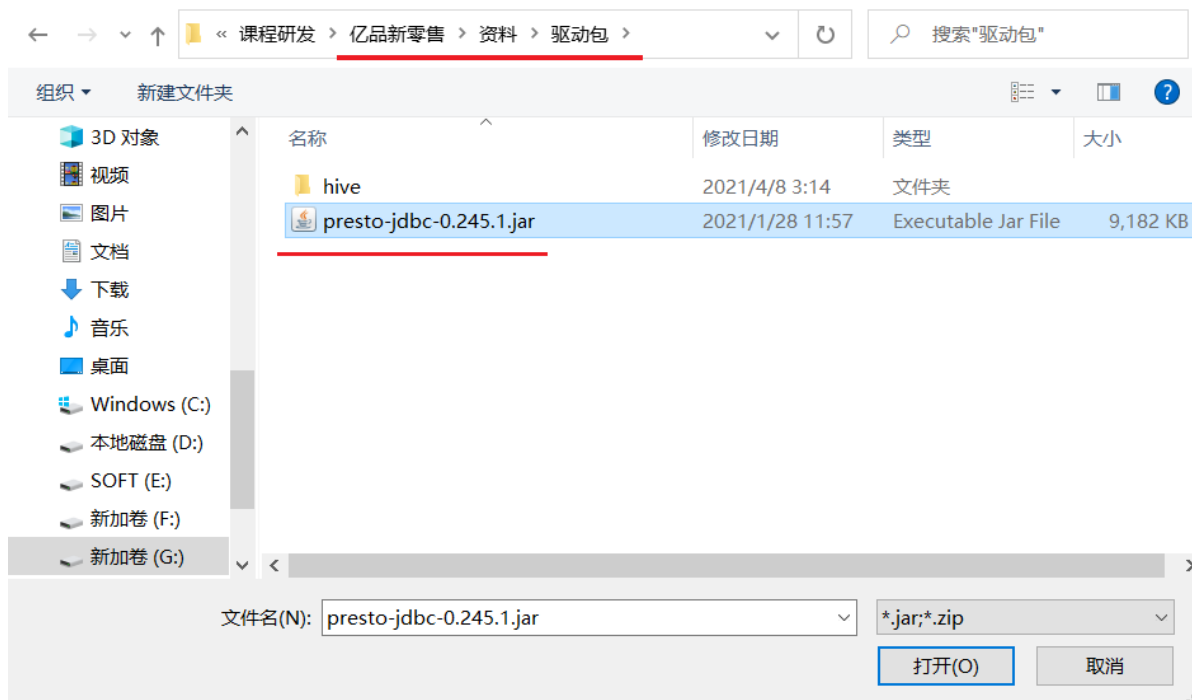
完成(F)

取消

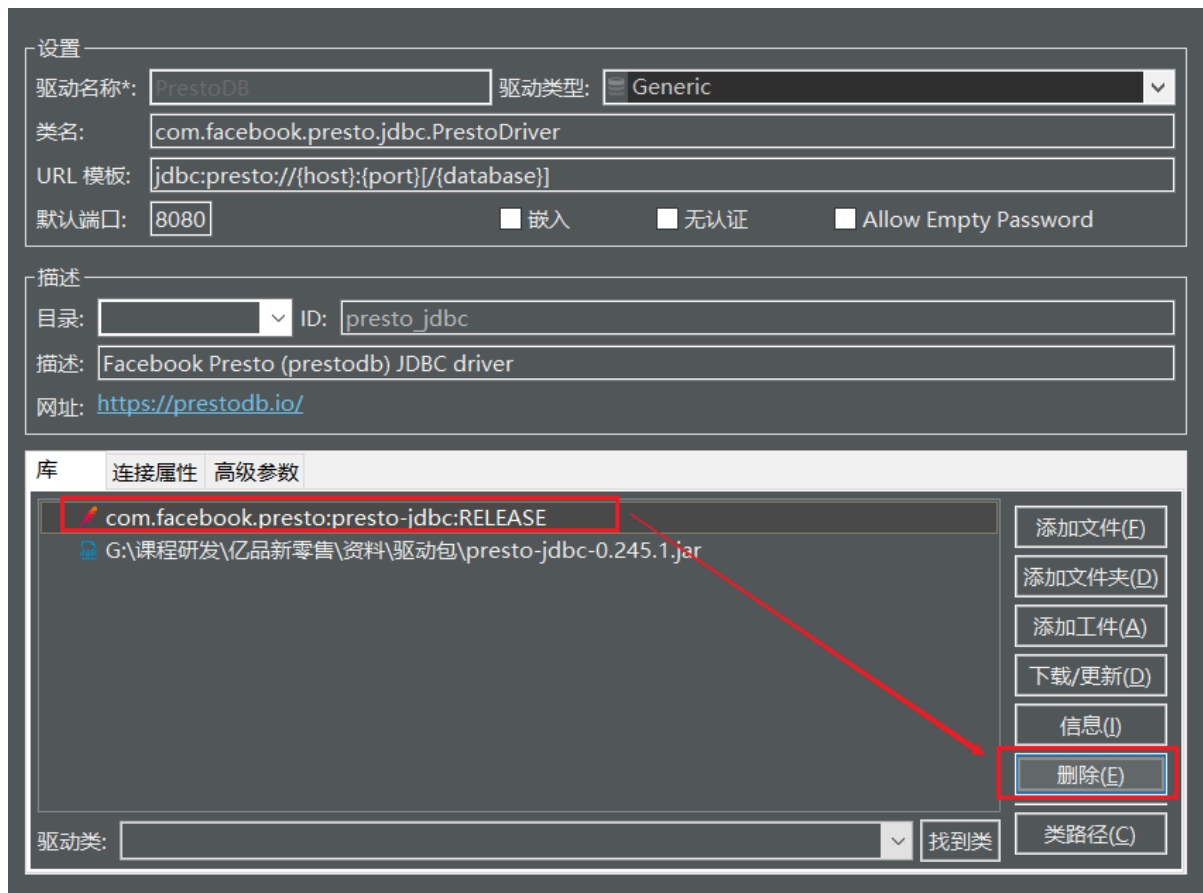
2. 点击【添加文件】



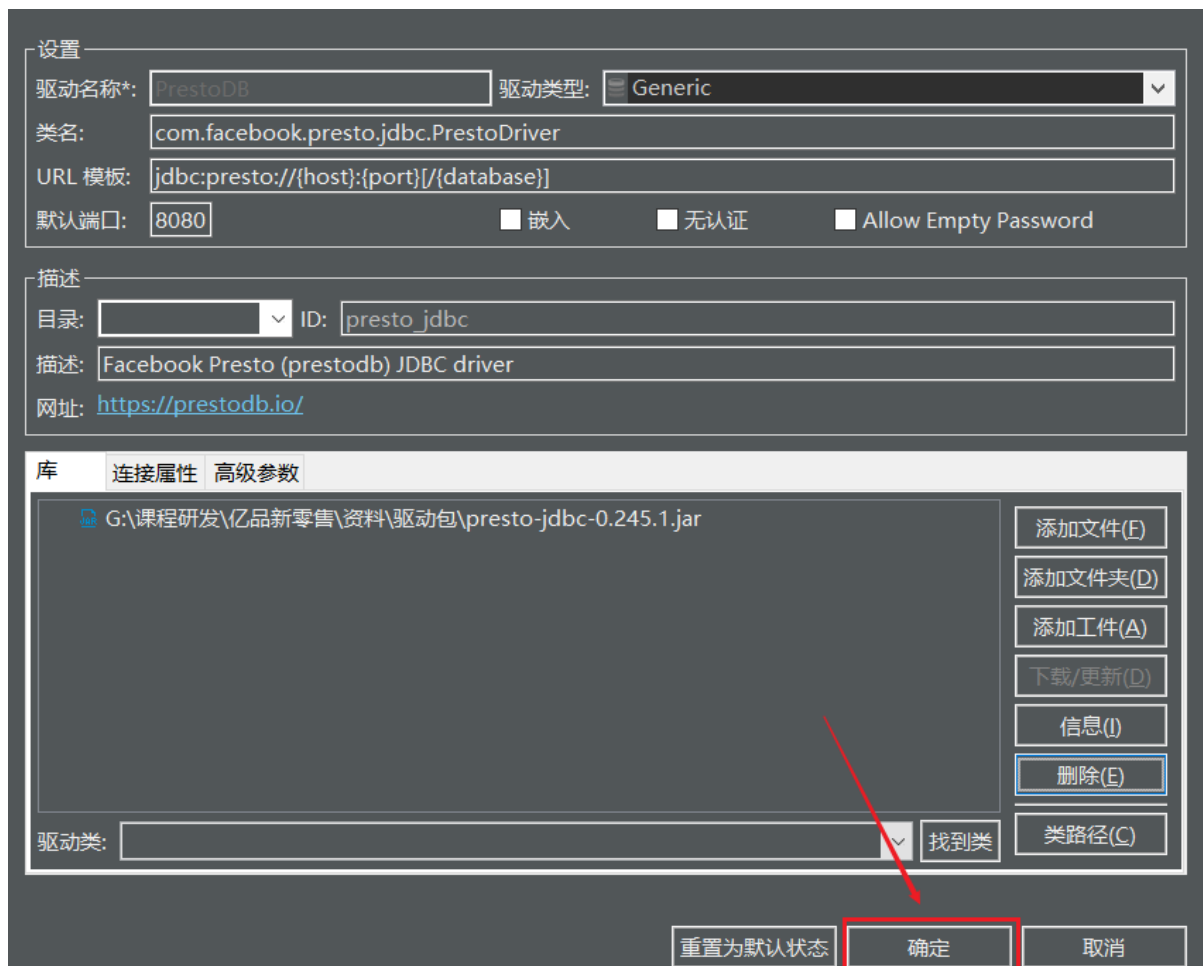
3. 选中提供好的驱动包，目录：【亿品新零售/资料/驱动包/presto-jdbc-0.245.1.jar】，【确定】



4. 删除自带的jar包链接：



5. 【确定】



设置

驱动名称*: 驱动类型:

类名:

URL 模板:

默认端口: ☐ 嵌入 ☐ 无认证 ☐ Allow Empty Password

描述

目录: ID:

描述:

网址:

库 连接属性 高级参数

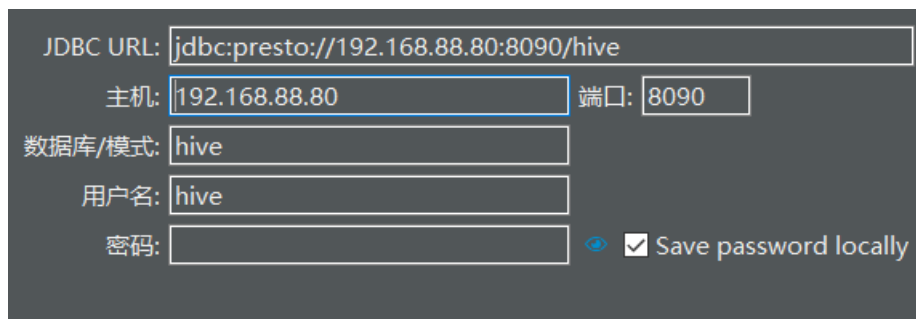
添加文件(E) 添加文件夹(D) 添加工件(A) 下载/更新(D) 信息(I) 删除(E) 类路径(C)

驱动类: 找到类

重置为默认状态 确定 取消

9.2.3 配置连接地址

1. 填写对应的【主机】、【端口】、【数据库/模式】、【用户名】



JDBC URL:

主机: 端口:

数据库/模式:

用户名:

密码:

☒ Save password locally

2. 点击【测试链接】



连接设置

初始化

Shell 命令

客户端认证

常规

元数据

错误处理

结果集

编辑器

数据格式化

展示

SQL 编辑器

SQL 处理

常规

驱动属性

SSH

Proxy

JDBC URL: jdbc:presto://192.168.88.80:8090/hive

主机: 192.168.88.80 端口: 8090

数据库/模式: hive

用户名: hive

密码:

☒ Save password locally

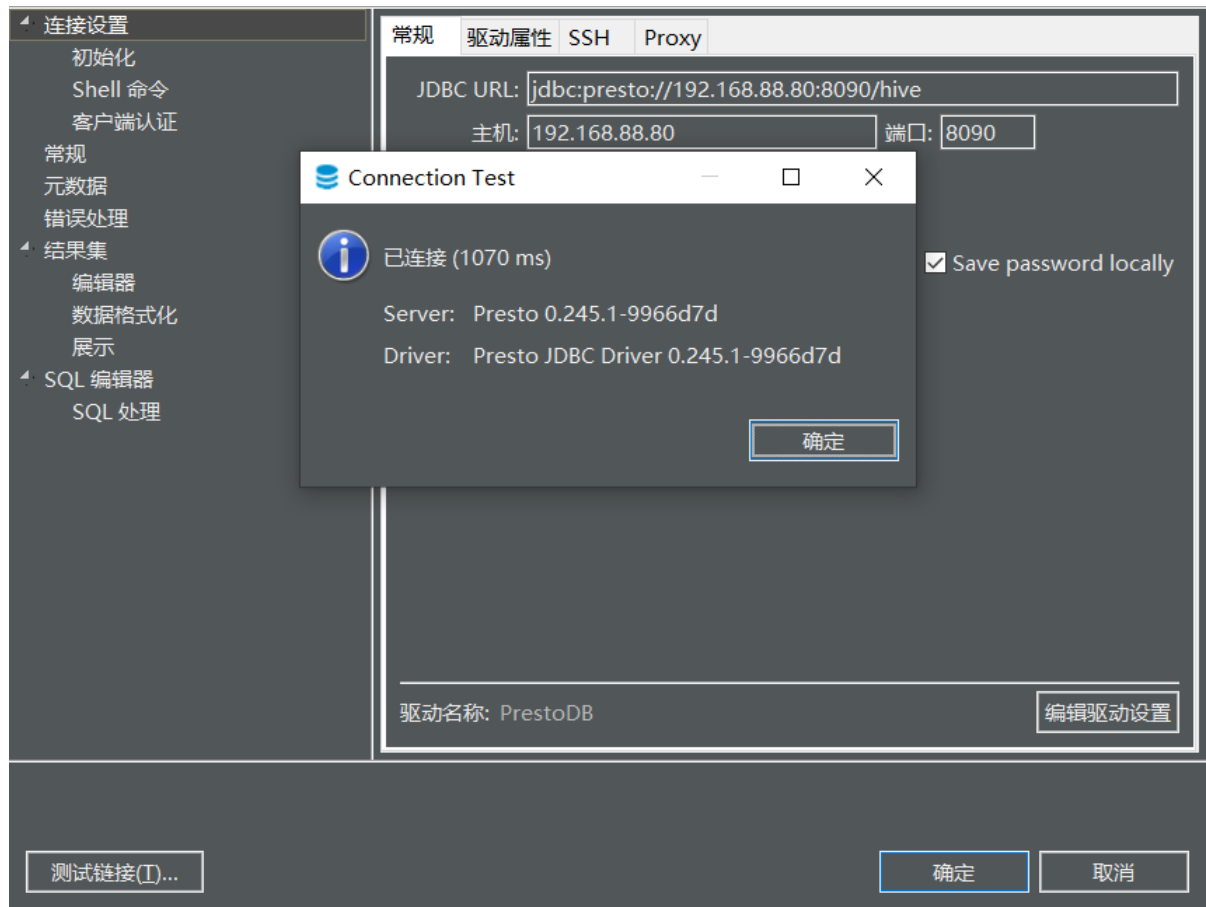
驱动名称: PrestoDB

编辑驱动设置

测试链接(T)...

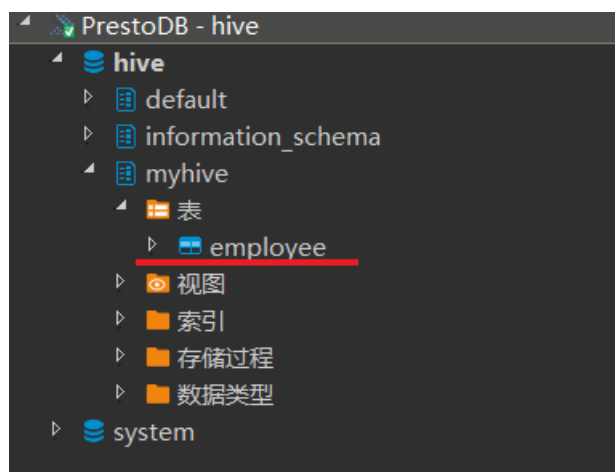
确定

取消

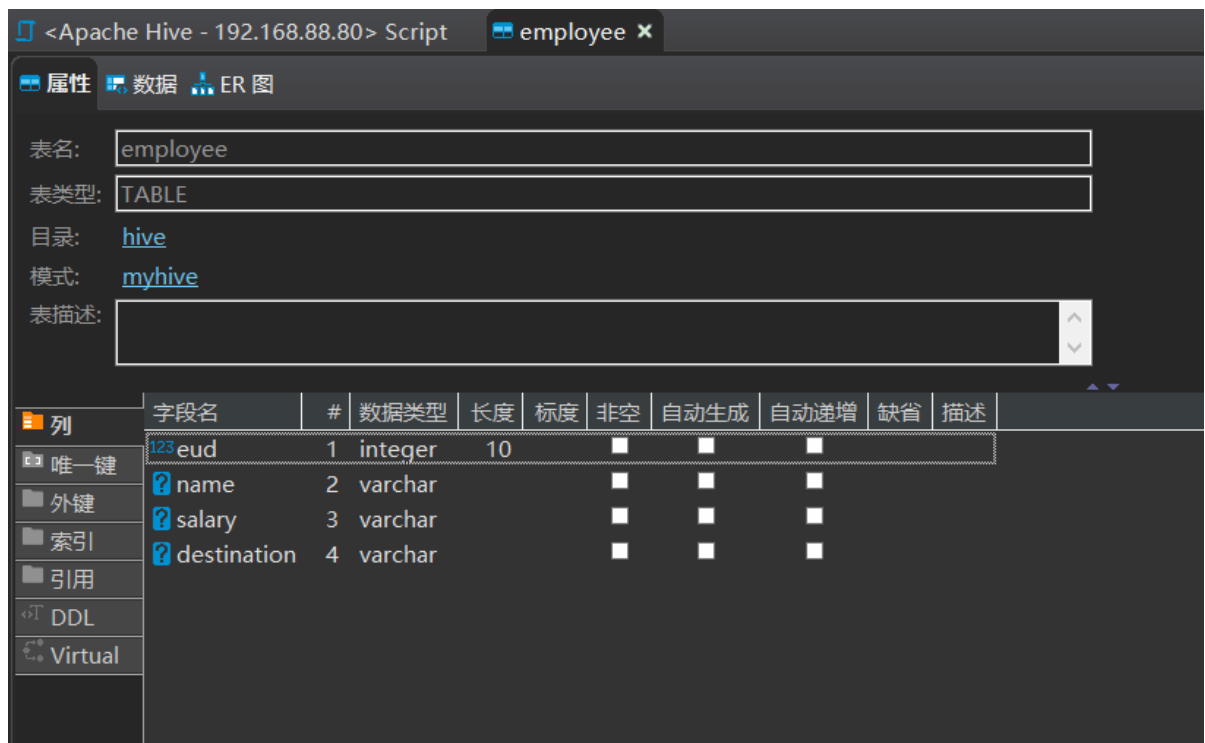


9.2.4 测试

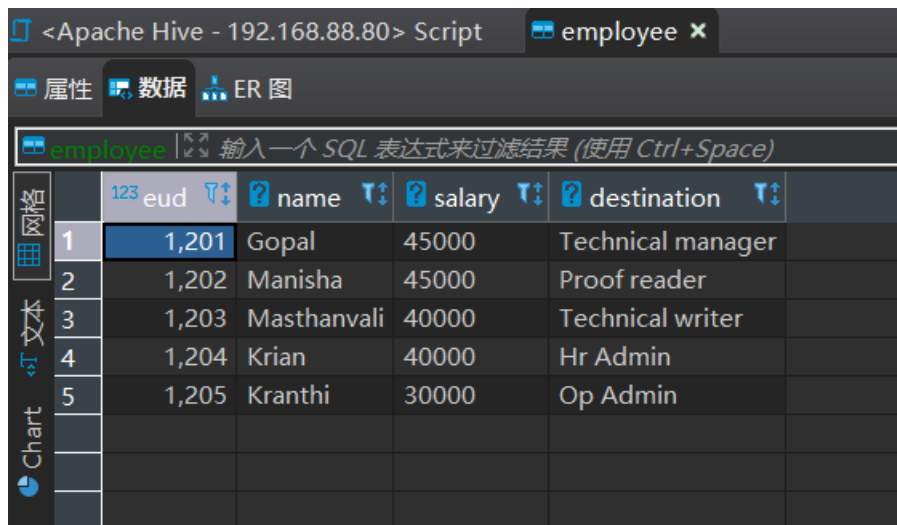
1. 展开Presto连接菜单



2. 双击表名，查看详细属性

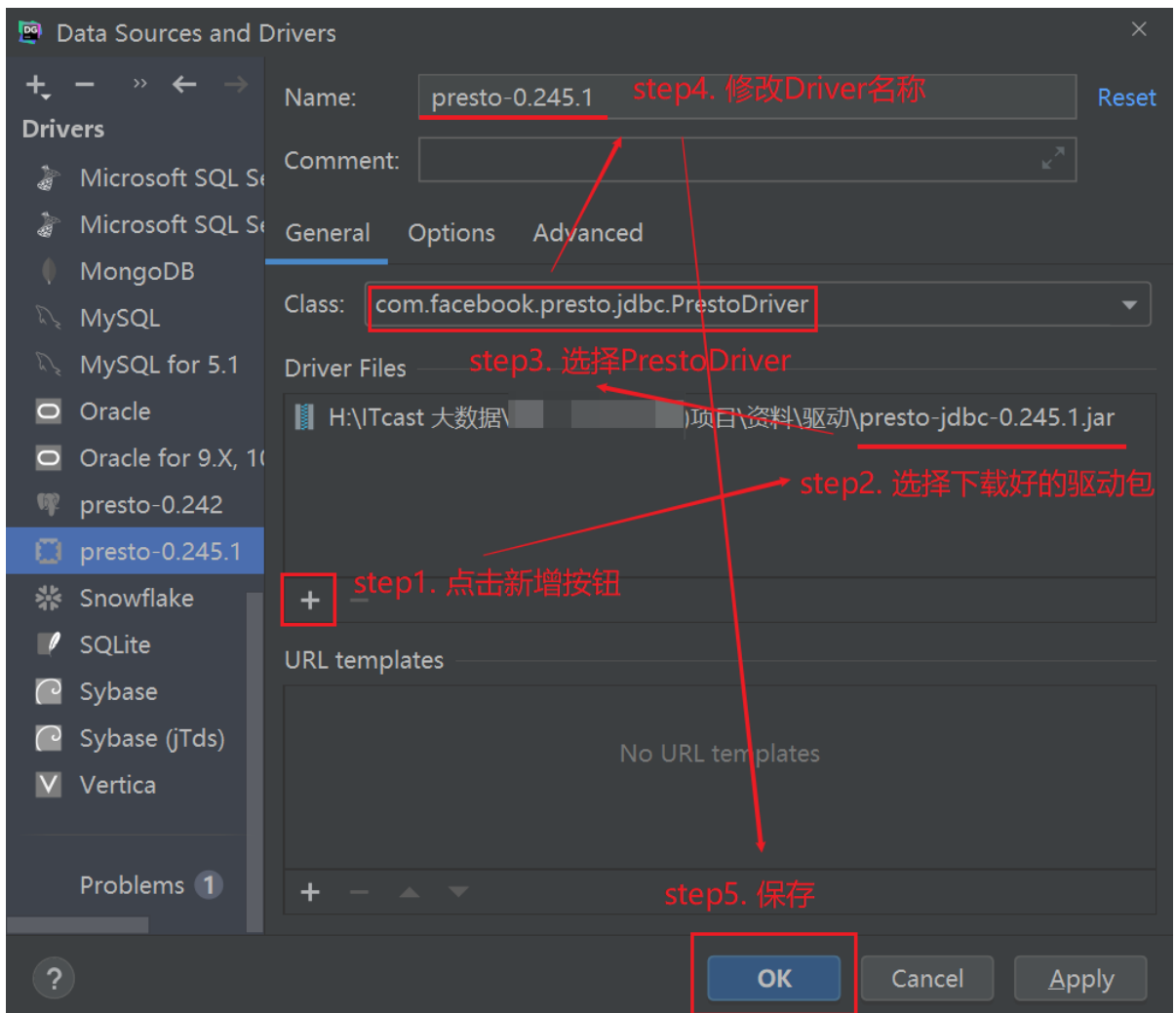
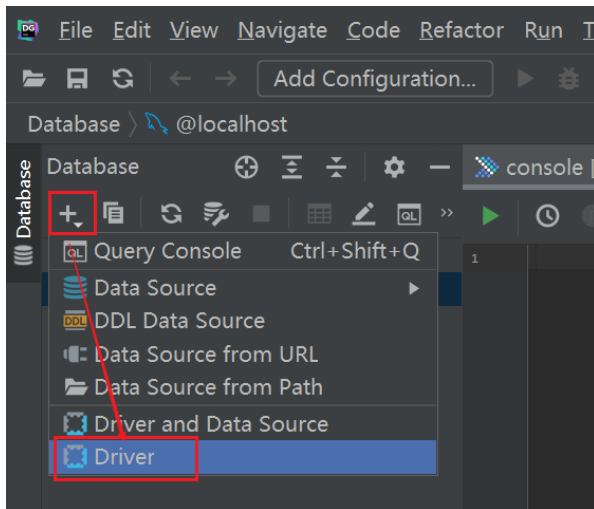


3. 点击【数据】TAB

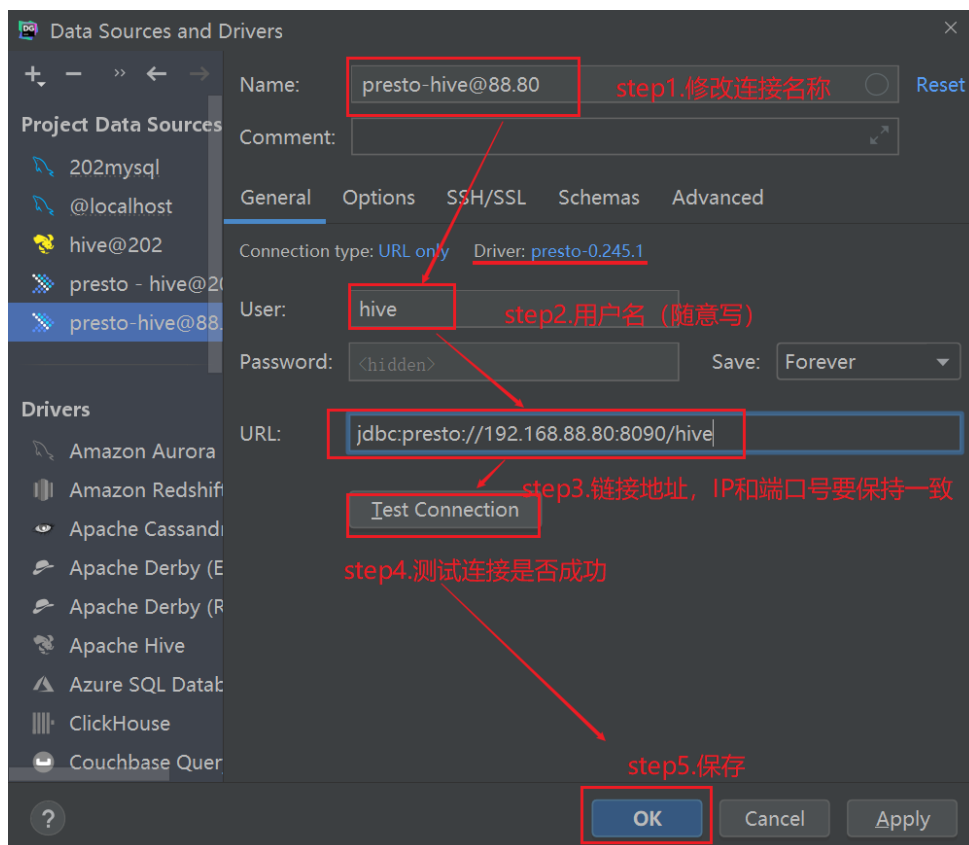
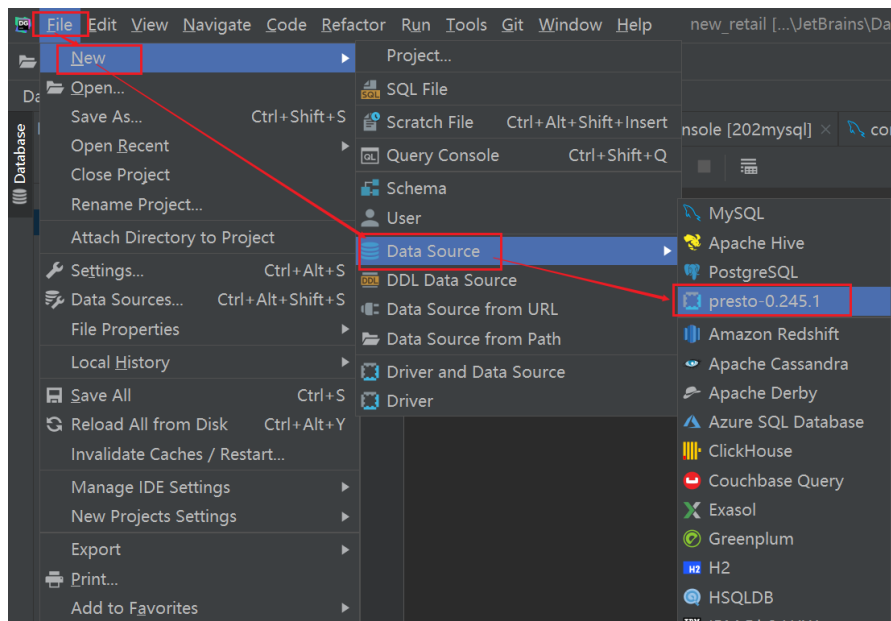


9.3 DataGrip客户端

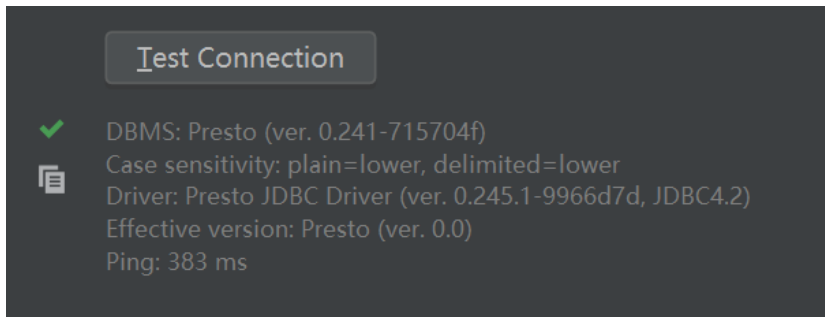
9.3.1 创建Driver



9.3.2 创建连接

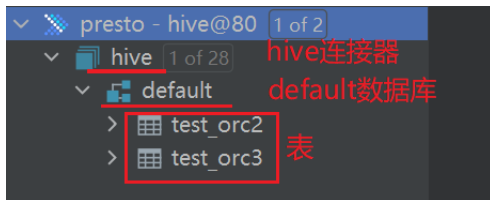


jdbc:presto://192.168.88.80:8090/hive

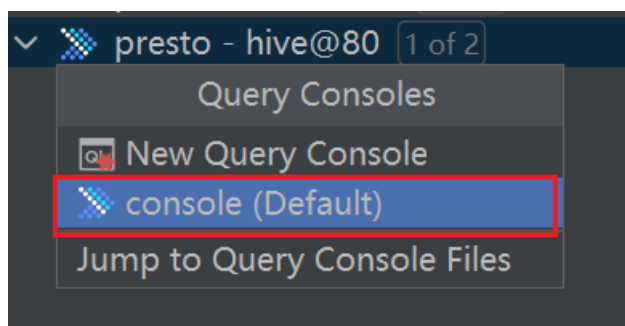
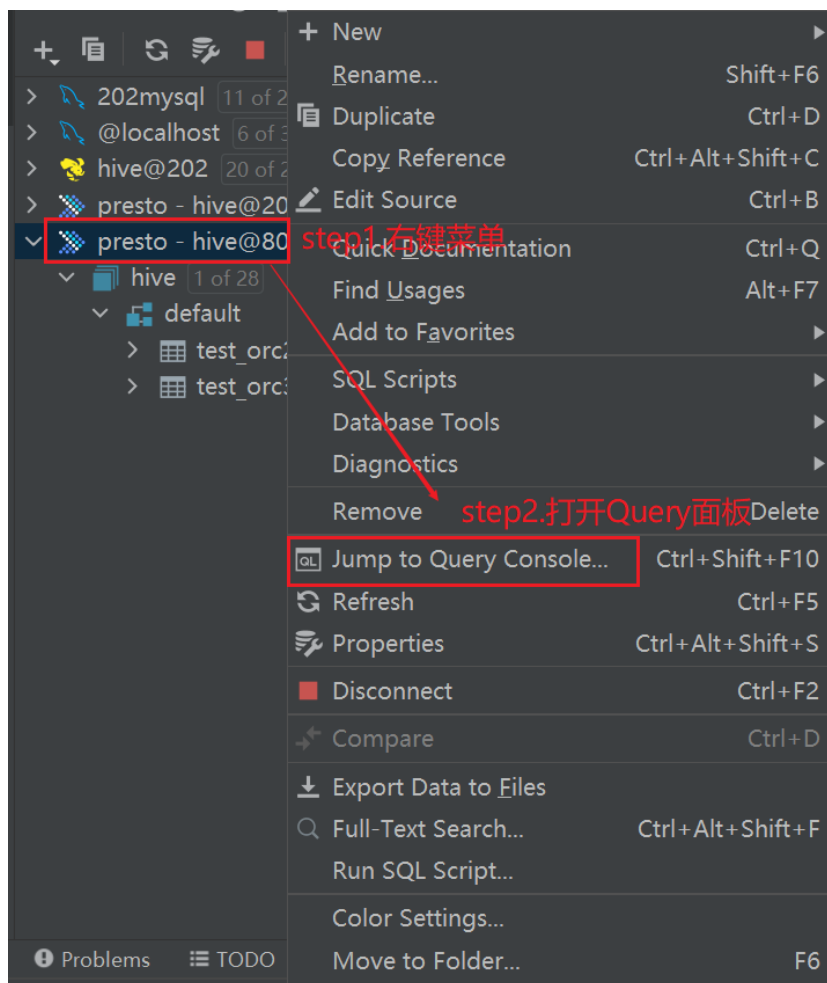


9.3.3 测试

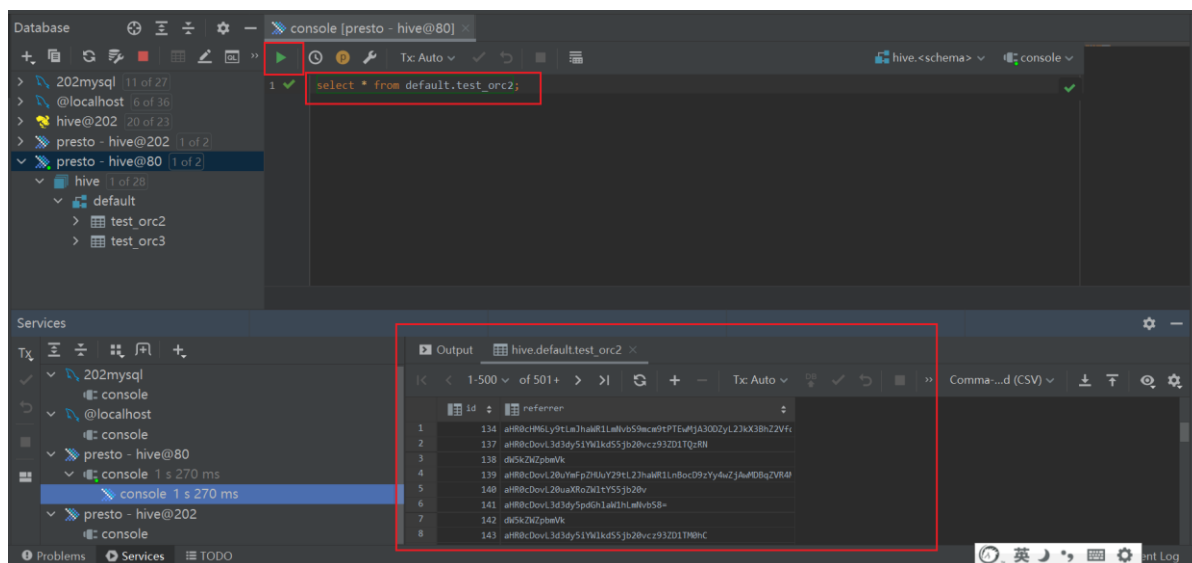
9.3.3.1 DB菜单栏



9.3.3.2 打开Query面板



9.3.3.3 执行SQL



10. 日期与时间类型

10.1 日期与时间值

DATE '2001-08-22'

TIMESTAMP '2001-08-22 03:04:05.321'

10.2 时间和字符串转换

10.2.1 时间转字符

`date_format(timestamp, format) → varchar`

将timestamp转换化为指定format格式的string。

10.2.2 字符转时间

`date_parse(string, format) → timestamp`

将format格式的string转换为时间类型。

10.2.3 字符转日期

`date(x) → date`

等同于`CAST(x AS date)`

10.2.4 format格式

年：%Y

月：%m

日：%d

时：%H

分：%i

秒：%s

周几：%w(0..6)

10.2.5 时间加减

`date_add(unit, value, timestamp) → [same as input]`

进行unit单位的时间运算。减法可以用负数来执行。

`date_diff(unit, timestamp1, timestamp2) → bigint`

时间timestamp2-timestamp1后，以unit单位进行展示差值。

11. Presto优化

11.1 数据存储格式

1) 合理设置分区

与Hive类似，Presto会根据元信息读取分区数据，合理的分区能减少Presto数据读取量，提升查询性能。

2) 使用列式存储

Presto对ORC文件读取做了特定优化，因此在Hive中创建Presto使用的表时，建议采用ORC格

式存储。相对于Parquet，Presto对ORC支持更好。

Parquet和ORC一样都支持列式存储，但是Presto对ORC支持更好，而Impala对Parquet支持更好。在数仓设计时，要根据后续可能的查询引擎合理设置数据存储格式。

3) 使用压缩

数据压缩可以减少节点间数据传输对IO带宽压力，对于即席查询需要快速解压，建议采用Snappy压缩。

4) 预先排序

对于已经排序的数据，在查询的数据过滤阶段，ORC格式支持跳过读取不必要的数据。比如对于经常需要过滤的字段可以预先排序。

```
INSERT INTO table nation_orc partition(p) SELECT * FROM nation SORT BY n_name;
```

如果需要过滤n_name字段，则性能将提升。

```
SELECT count(*) FROM nation_orc WHERE n_name='AUSTRALIA';
```

11.2 SQL优化

1) 只选择使用必要的字段

由于采用列式存储，选择需要的字段可加快字段的读取、减少数据量。避免采用*读取所有字段。

```
[✓]: SELECT time,user,host FROM tbl
```

```
[✗]: SELECT * FROM tbl
```

2) 过滤条件必须加上分区字段

对于有分区的表，where语句中优先使用分区字段进行过滤。acct_day是分区字段，visit_time是具体访问时间。

```
[✓]: SELECT time, user, host FROM tbl where acct_day=20171101
```

```
[✗]: SELECT * FROM tbl where visit_time=20171101
```


3) Group By语句优化

合理安排Group by语句中字段顺序对性能有一定提升。将Group By语句中字段按照每个字段distinct数据多少进行降序排列。

[✓]: SELECT GROUP BY uid, gender
[✗]: SELECT GROUP BY gender, uid

4) Order by时使用Limit

Order by需要扫描数据到单个worker节点进行排序，导致单个worker需要大量内存。如果是查询Top N或者Bottom N，使用limit可减少排序计算和内存压力。

[✓]: SELECT * FROM tbl ORDER BY time LIMIT 100
[✗]: SELECT * FROM tbl ORDER BY time

5) 用regexp_like代替多个like语句

Presto查询优化器没有对多个like语句进行优化，使用regexp_like对性能有较大提升

[✓]
SELECT
...
FROM
access
WHERE
regexp_like(method, 'GET|POST|PUT|DELETE')

[✗]
SELECT
...
FROM
access
WHERE
method LIKE '%GET%' OR
method LIKE '%POST%' OR
method LIKE '%PUT%' OR
method LIKE '%DELETE%'

6) 使用Join语句时将大表放在左边

Presto中join的默认算法是broadcast join，即将join左边的表分割到多个worker，然后将join右边的表数据整个复制一份发送到每个worker进行计算。如果右边的表数据量太大，则可能会报内存溢出错误。

[✓] SELECT ... FROM large_table l join small_table s on l.id = s.id

[✗] SELECT ... FROM small_table s join large_table l on l.id = s.id

7) 使用Rank函数代替row_number函数来获取TopN

在进行一些分组排序场景时，使用rank函数性能更好。

[✓]

```
SELECT checksum(rnk)
FROM (
    SELECT rank() OVER (PARTITION BY l_orderkey, l_partkey ORDER BY l_shipdate DESC) AS rnk
    FROM lineitem
) t
WHERE rnk = 1
```

[✗]

```
SELECT checksum(rnk)
FROM (
    SELECT row_number() OVER (PARTITION BY l_orderkey, l_partkey ORDER BY l_shipdate DESC) AS rnk
    FROM lineitem
) t
WHERE rnk = 1
```

11.3 替换非ORC的Hive表

如果之前的hive表没有用到ORC和snappy，那么怎么无缝替换而不影响线上的应用：

比如如下一个hive表：

```
CREATE TABLE bdc_dm.res_category(  
channel_id1 int comment '1级渠道id',  
province string COMMENT '省',  
city string comment '市',  
uv int comment 'uv'  
)  
comment 'example'  
partitioned by (landing_date int COMMENT '日期:yyyymmdd')  
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' COLLECTION ITEMS TERMINATED BY ',' MAP KEYS TERMINATED BY ':' LINES TERMINATED BY '\n';
```

建立对应的orc表

```
CREATE TABLE bdc_dm.res_category_orc(  
channel_id1 int comment '1级渠道id',  
province string COMMENT '省',  
city string comment '市',  
uv int comment 'uv'  
)  
comment 'example'  
partitioned by (landing_date int COMMENT '日期:yyyymmdd')  
row format delimited fields terminated by '\t'  
stored as orc  
TBLPROPERTIES ("orc.compress"="SNAPPY");
```

先将数据灌入orc表，然后更换表名

```
insert overwrite table bdc_dm.res_category_orc partition(landing_date)  
select * from bdc_dm.res_category where landing_date >= 20171001;  
  
ALTER TABLE bdc_dm.res_category RENAME TO bdc_dm.res_category_tmp;  
ALTER TABLE bdc_dm.res_category_orc RENAME TO bdc_dm.res_category;
```

其中res_category_tmp是一个备份表，若线上运行一段时间后没有出现问题，则可以删除该表。

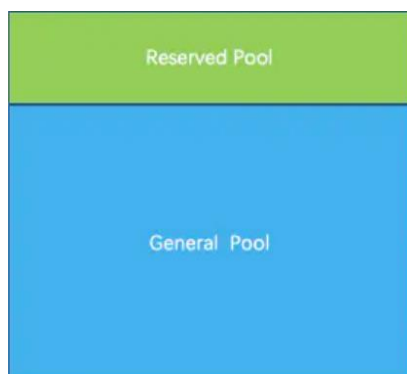
11.4 内存调优

11.4.1 内存管理原理

11.4.1.1 User & System memory

Presto把自己管理的内存分为两大类：user memory和system memory，所谓的user memory是跟用户数据相关的，比如读取用户输入数据会占据相应的内存，这种内存的占用量跟用户底层数据量大小是强相关的；system memory则是执行过程中衍生出的副产品，比如tablescan表扫描，write buffers写入缓冲区，跟查询输入的数据本身不强相关的内存。

11.4.1.2 内存池



执行过程中，presto是从具体的内存池中来实现分配user memory和system memory的。Presto有两种内存池，分别为常规内存池GENERAL_POOL、预留内存池RESERVED_POOL。

注：0.201之后SYSTEM_POOL已经被废弃，GENERAL_POOL扮演了之前GENERAL_POOL及SYSTEM_POOL的作用，提供user memory和system memory。

1. GENERAL_POOL：在一般情况下，一个查询执行所需要的user/system内存都是从general pool中分配的，reserved pool在一般情况下是空闲不用的。

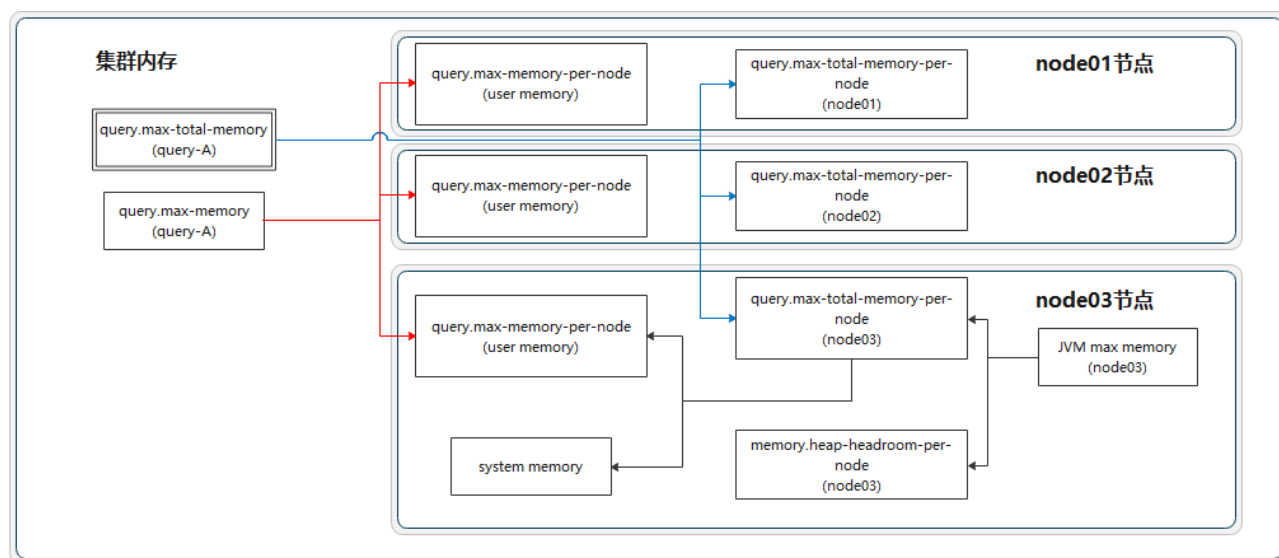
2. RESERVED_POOL：大部分时间里是不参与计算的，但是当集群中某个Worker节点的general pool消耗殆尽之后，coordinator会选择集群中内存占用最多的查询，把这个查询分配到reserved pool，这样这个大查询自己可以继续执行，而腾出来的内存也使得其它的查询可以继续执行，从而避免整个系统阻塞。

reserved pool到底多大呢？这个是没有直接的配置可以设置的，他的大小上限就是集群允许的最大的查询的大小(query.total-max-memory-per-node)。

reserved pool也有缺点，一个是在普通模式下这块内存会被浪费掉了，二是大查询可以用Hive

来替代。因此也可以禁用掉reserved pool (experimental.reserved-pool-enabled=false)，那系统内存耗尽的时候没有reserved pool怎么办呢？它有一个OOM Killer的机制，对于超出内存限制的大查询SQL将会被系统Kill掉，从而避免影响整个presto。

11.4.2 相关参数



query.max-memory-per-node：单个query操作在单个worker上user memory能用的最大值

query.max-total-memory-per-node：单个query操作可在单个worker上使用的最大(user + system)内存量

query.max-memory：单个query在整个集群中允许占用的最大user memory

query.max-total-memory: 单个query在整个集群中允许占用的最大(user + system) memory

当这些阈值被突破的时候，query会以insufficient memory的错误被终结。

在高内存压力下保持系统稳定的另一种机制是协作阻止机制。当general pool内存池已满时，operator会被置为blocked阻塞状态，直到通用池中的内存可用为止。此机制可防止激进的查询填满JVM堆并引起可靠性问题。

memory.heap-headroom-per-node：这个内存是JVM堆中预留给第三方库的内存分配，presto无法跟踪统计，默认值是-Xmx * 0.3

GeneralPool = 服务器总内存 - ReservedPool - memory.heap-headroom-per-node - Linux系统内存

11.4.3 内存调优

Presto由于是完全基于内存的计算，经常出现OOM，需要调整内存。

1、Query exceeded per-node total memory limit of xx

适当增加query.max-total-memory-per-node。

2、Query exceeded distributed user memory limit of xx

适当增加query.max-memory。

3、Could not communicate with the remote task. The node may have crashed or be under too much load

内存不够，导致节点crash，可以查看/var/log/message。

query.max-memory-per-node和query.max-total-memory-per-node是query操作使用的主要内存配置，因此这两个配置可以适当加大。而memory.heap-headroom-per-node是三方库的内存，默认值是-Xmx * 0.3，可以手动改小一些。

假如某个Worker有64G内存，可以只给headroom分配8G，max-total-memory-per-node（单个查询在单个机器user+system）分配16G，另外给Linux操作系统留下16G。

那么ReservedPool上限 = query.max-total-memory-per-node = 16G；GeneralPool = 64G - 16G - 8G - 16G = 24G。

GeneralPool = 服务器总内存 - ReservedPool - memory.heap-headroom-per-node - Linux系统内存。

当然上面给的只是一个思路，列出了一些需要考虑的因素，具体的参数应该设置多大还是要根据实际工作负载来进行压力测试。

注意：

1、query.max-memory-per-node < query.max-total-memory-per-node。

2、query.max-memory < query.max-total-memory。

3、query.max-total-memory-per-node 与memory.heap-headroom-per-node 之和必须小于 jvm max memory，也就是jvm.config 中配置的-Xmx。