



Hive简介



摘要

2

- Hive基本原理
- Hive基本操作



摘要

3

- Hive基本原理
- Hive基本操作

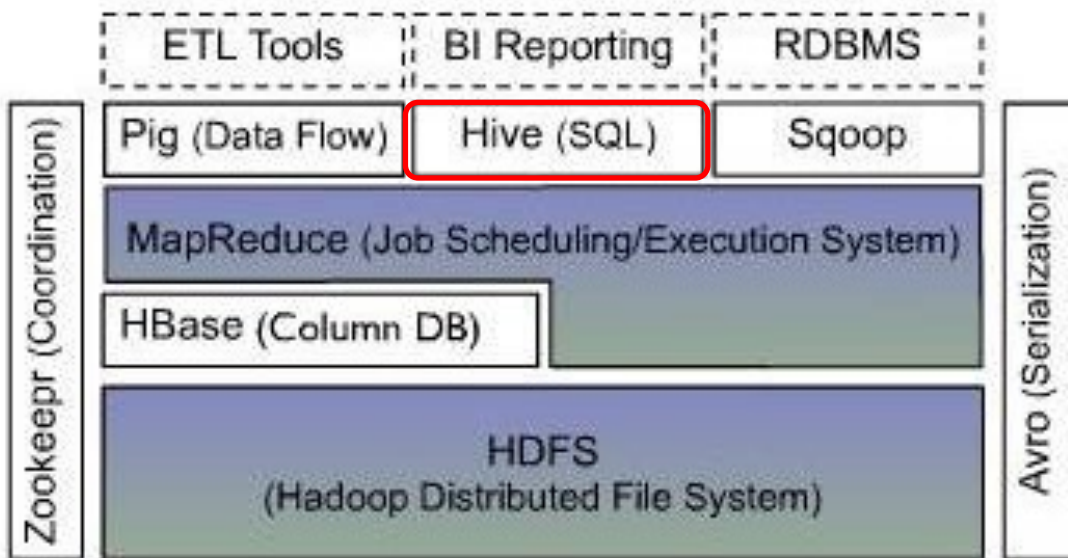


Hadoop生态

4



<http://hive.apache.org>



本讲义先介绍Hive 2.x版本，后对比3.x/4.x版本的变化



使用Hadoop进行数据分析

- 通过前面的课程知道，很多分析任务可以通过**Hadoop**集群进行，即可以通过**Hadoop**集群将任务分布到数百甚至上千个节点中进行分析，通过并行执行缩短分析的时间
- **Hadoop**通过**MapReduce**的并行化方式进行并行处理，能够充分利用数目庞大的服务器节点
- 但是，**MapReduce**是一个底层的编程接口，对于数据分析人员来说，这个编程接口并不是十分友好，还需要进行大量的编程以及调试工作



在Hadoop上加入数据分析功能

- 显然，为了能够支持一个类似于**SQL**相关的数据查询语言，**Hadoop**还需要加入一些额外的模块才能够方便数据分析人员的使用，这些额外的模块包括：
 - ▣ 数据查询语言本身的定义与构造，这是与终端用户进行交互的接口，最简单的可以通过命令行接口的方式展开用户与系统的交互
 - ▣ 构造数据查询语言的执行引擎，即将上述的查询语言进行编译，并通过分布式的执行引擎完成查询，在**Hive**中，执行引擎会将查询语言翻译为多个**MapReduce**的任务序列，交给**MapReduce**程序去执行
 - ▣ 数据查询语言本身需要定义一套数据组织的格式



Hive是什么

7

- **Hive**是一个基于**Apache Hadoop**的**数据仓库**。对于数据存储与处理，**Hadoop**提供了主要的扩展和容错能力。
- **Hive**设计的初衷：对于大量的数据，使得数据汇总、查询和分析更加简单。它提供了**SQL**，允许用户简单地进行查询、汇总和数据分析。
- **Hive**的**SQL**给予了用户多种方式来集成自己的功能，然后做定制化的查询，例如用户自定义函数。



Hive 优点

8

- **可扩展性**：横向扩展，**Hive** 可以自由的扩展集群的规模，一般情况下不需要重启服务。
- **延展性**：**Hive** 支持自定义函数，用户可以根据自己的需求来实现自己的函数。
- **良好的容错性**：可以保障即使有节点出现问题，**SQL** 语句仍可完成执行。



Hive 缺点

9

- **Hive** 不支持记录级别的增删改操作，但是用户可以通过查询生成新表或者将查询结果导入到文件中（新版本支持记录级别的插入操作）
- **Hive** 的查询延时很严重，因为 **MapReduce Job** 的启动过程消耗很长时间，所以不能用在交互查询系统中。
- **Hive** 不是为在线事务处理而设计，所以主要用来做 **OLAP**（联机分析处理），而不是 **OLTP**（联机事务处理）。它最适合用于传统的数据仓库任务。



RDBMS vs Hive

10

- **Hive**和关系数据库存储文件的系统不同，**Hive**使用的是**Hadoop**的**HDFS**，关系数据库则是服务器本地的文件系统；
- **Hive**使用的计算模型是**MapReduce**，而关系数据库则是自己设计的计算模型；
- 关系数据库都是为实时查询的业务进行设计的，而**Hive**则是为海量数据做数据挖掘设计的，实时性很差；实时性的区别导致**Hive**的应用场景和关系数据库有很大的不同；
- **Hive**很容易扩展自己的存储能力和计算能力，这个是继承**Hadoop**的，而关系数据库在这个方面要比数据库差很多。



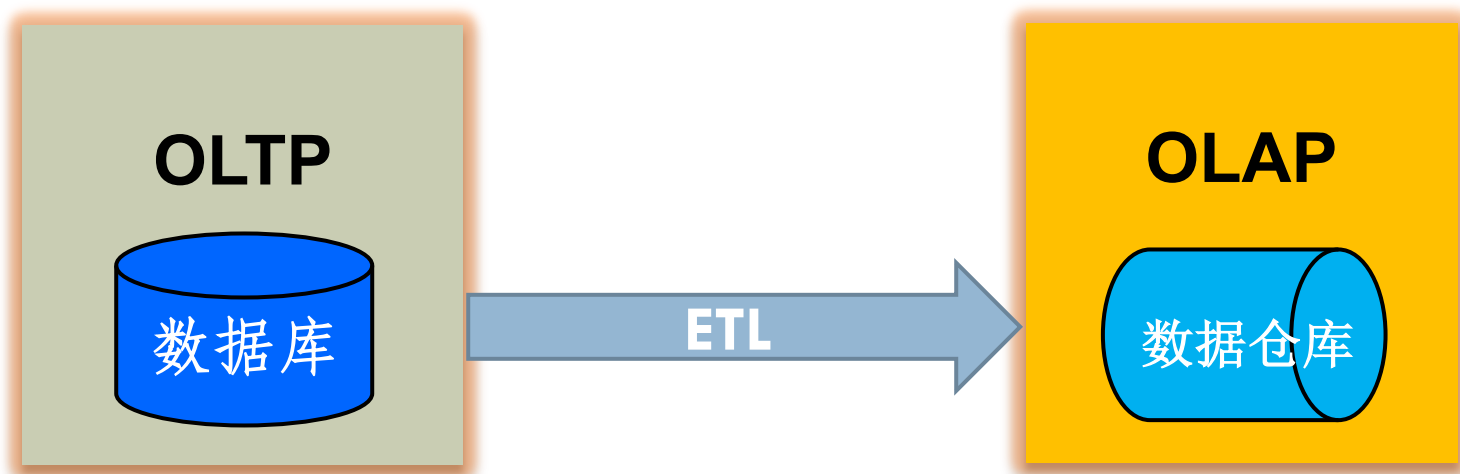
RDBMS vs Hive

对比项	Hive	RDBMS
查询语言	HQL	SQL
数据存储	HDFS	Raw Device or Local FS
执行器	MapReduce	Executor
数据插入	支持批量导入/单条插入	支持单条或者批量导入
数据操作	覆盖追加	行级更新删除
处理数据规模	大	小
执行延迟	高	低
分区	支持	支持
索引	0.8 版本之后加入简单索引	支持复杂的索引
扩展性	高（好）	有限（差）
数据加载模式	读时模式（快）	写时模式（慢）
应用场景	海量数据查询	实时查询

Hive 具有 SQL 数据库的外表，但应用场景完全不同，Hive 只适合用来做海量离线数据统计分析，也就是数据仓库。

HBase vs Hive

- **HBase**和**Hive**在大数据架构中处在不同位置，**HBase**主要解决实时数据查询问题，**Hive**主要解决数据处理和计算问题，一般是配合使用。
 - ▣ **HBase**主要针对的是OLTP应用：强调实时性和随机访问
 - ▣ **Hive**主要针对的是OLAP应用：强调批处理和离线数据分析





HBase vs Hive

13

区别

- **HBase**: 基于Hadoop的NoSQL数据库，主要适用于海量明细数据（十亿、百亿）的随机实时查询，如日志明细、交易清单、轨迹行为等。
- **Hive**: 基于Hadoop的数据仓库，严格来说，不是数据库，主要是让开发人员能够通过SQL来计算和处理HDFS上的结构化数据，适用于离线的批量数据计算。
 - 通过元数据来描述Hdfs上的结构化文本数据，通俗点来说，就是定义一张表来描述HDFS上的结构化文本，包括各列数据名称，数据类型是什么等，方便我们处理数据，当前很多SQL ON Hadoop的计算引擎均用的是hive的元数据，如Spark SQL、Impala等；
 - 通过SQL来处理 and 计算HDFS的数据，Hive将SQL翻译为MapReduce来处理数据。



HBase vs Hive

14

区别

- **Hive**中的表是纯逻辑表，就只是表的定义等，即表的元数据。**Hive**本身不存储数据，它完全依赖**HDFS**和**MapReduce**。这样就可以将结构化的数据文件映射为为一张数据库表，并提供完整的**SQL**查询功能，并将**SQL**语句最终转换为**MapReduce**任务运行。而**HBase**表是物理表，适合存放非结构化的数据。
- **Hive**是基于**MapReduce**来处理数据，而**MapReduce**处理数据是基于行的模式；**HBase**处理数据是基于列的而不是基于行的模式，适合海量数据的随机访问。



HBase vs Hive

15

区别

- **HBase**的表是疏松的存储的，因此用户可以给行定义各种不同的列；而**Hive**表是稠密型，即定义多少列，每一行有存储固定列数的数据。
- **Hive**使用**Hadoop**来分析处理数据，而**Hadoop**系统是批处理系统，因此不能保证处理的低延迟问题；而**HBase**是近实时系统，支持实时查询。
- **Hive**不提供row-level的更新，它适用于大量append-only数据集（如日志）的批任务处理。而基于**HBase**的查询，支持和row-level的更新。
- **Hive**提供完整的**SQL**实现，通常被用来做一些基于历史数据的挖掘、分析。而**HBase**不适用与有join，多级索引，表关系复杂的应用场景。



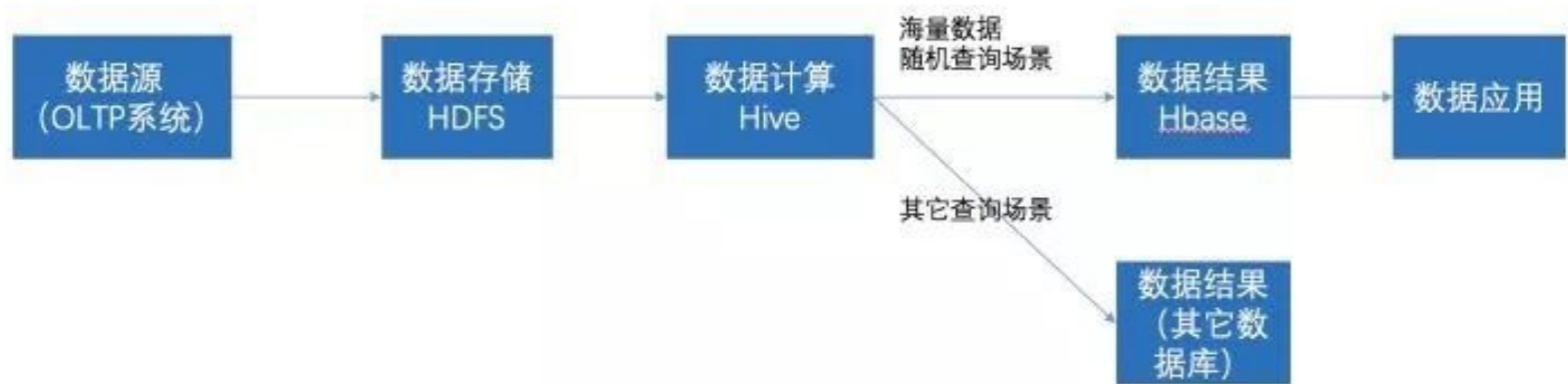
HBase vs Hive

16

□ 关系

▣ 在大数据架构中，两者是协作关系，数据流一般如下图：

- 通过ETL工具将数据源抽取到HDFS存储；
- 通过Hive清洗、处理和计算原始数据；
- Hive清洗处理后的结果，如果是面向海量数据随机查询场景的可存入HBase；
- 数据应用从HBase查询数据。





Hive的应用范围举例

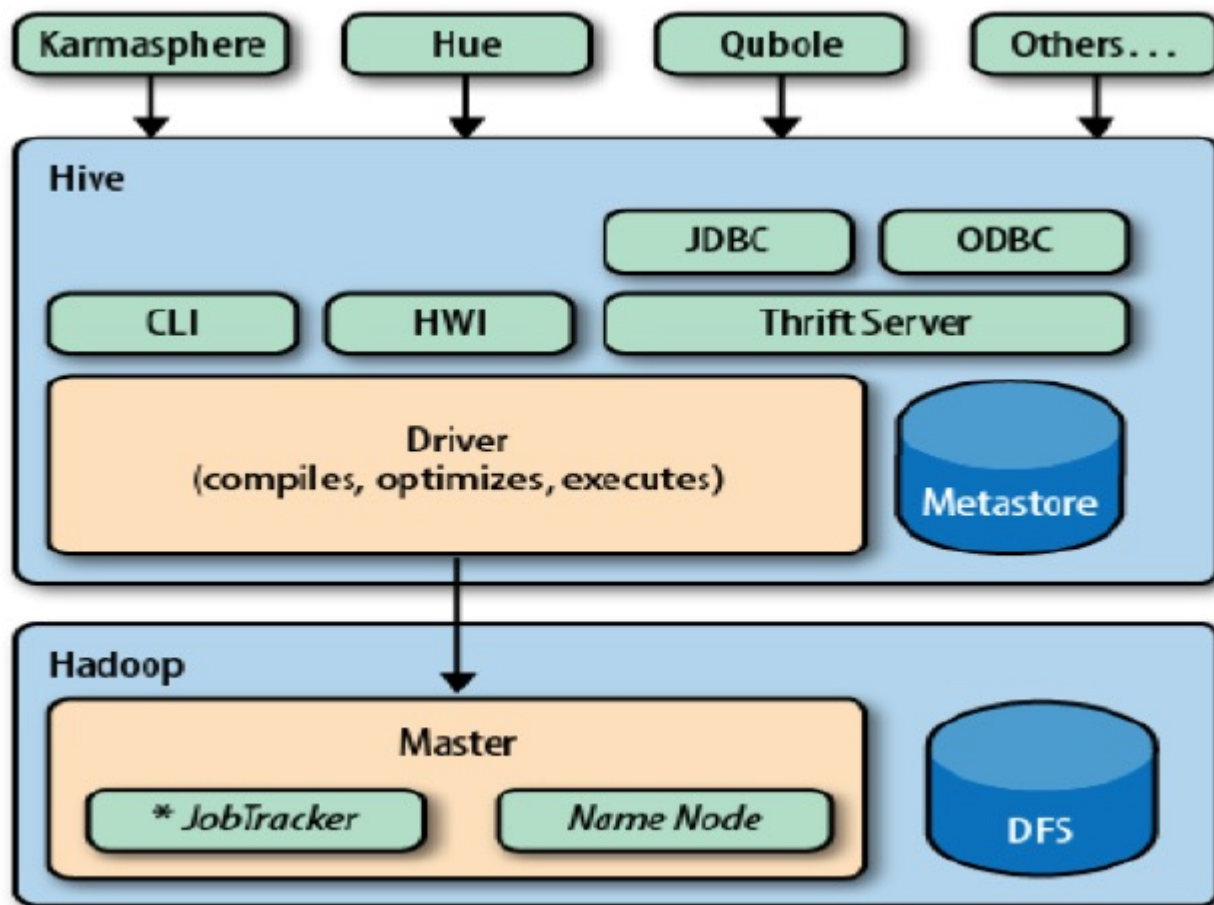
17

- **日志分析**：在互联网公司中，每天会产生大量的日志数据，分析这些日志数据是每天都需要进行的重要工作；日志分析可以优化系统，可以获知用户行为，也可以获知数据的统计信息
- **数据挖掘**：通过对结构化数据的挖掘，能够获得原先使用者没有意识到的信息
- **文档索引**：可以对一系列文档进行分析，并形成文档的索引结构，不一定是完整的倒排表，可能是关联信息的索引
- **商业智能信息处理**：可以对商业信息进行查询分析，从中可以获得一些智能决策的信息
- **即时查询以及数据验证**：数据分析人员可能临时需要验证数据的特性，需要查询引擎迅速进行数据统计分析



Hive的体系结构

18





Hive的组成模块(1)

19

- **Driver**组件：核心组件，整个Hive的核心。该组件包括**Compiler**、**Optimizer**和**Executor**，它的作用是将我们写的HQL语句进行解析、编译优化，生成执行计划，然后调用底层的**MapReduce**计算框架。
 - ▣ **Compiler**：Hive需要一个编译器，将HiveQL语言编译成中间表示，包括对于HiveQL语言的分析，执行计划的生成等工作
 - ▣ **Optimizer**：进行优化
 - ▣ **Executor**：执行引擎，在**Driver**的驱动下，具体完成执行操作，包括**MapReduce**执行，或者HDFS操作，或者元数据操作



Hive的组成模块(2)

- **Metastore**组件：数据服务组件，用以存储Hive的元数据：存储操作的数据对象的格式信息，在HDFS中的存储位置的信息以及其他用于数据转换的信息SerDe等。Hive的元数据存储关系数据库里，Hive支持的关系数据库有derby、mysql。
- **CLI**组件：Command Line Interface，命令行接口。
- **ThriftServers**：提供JDBC和ODBC接入的能力，它用来进行可扩展且跨语言的服务的开发，Hive集成了该服务，能让不同的编程语言调用Hive的接口。
- **Hive WEB Interface (HWI)**：Hive客户端提供了一种通过网页的方式访问hive所提供的服务。这个接口对应Hive的HWI组件（hive web interface）



Hive的组成模块(3)

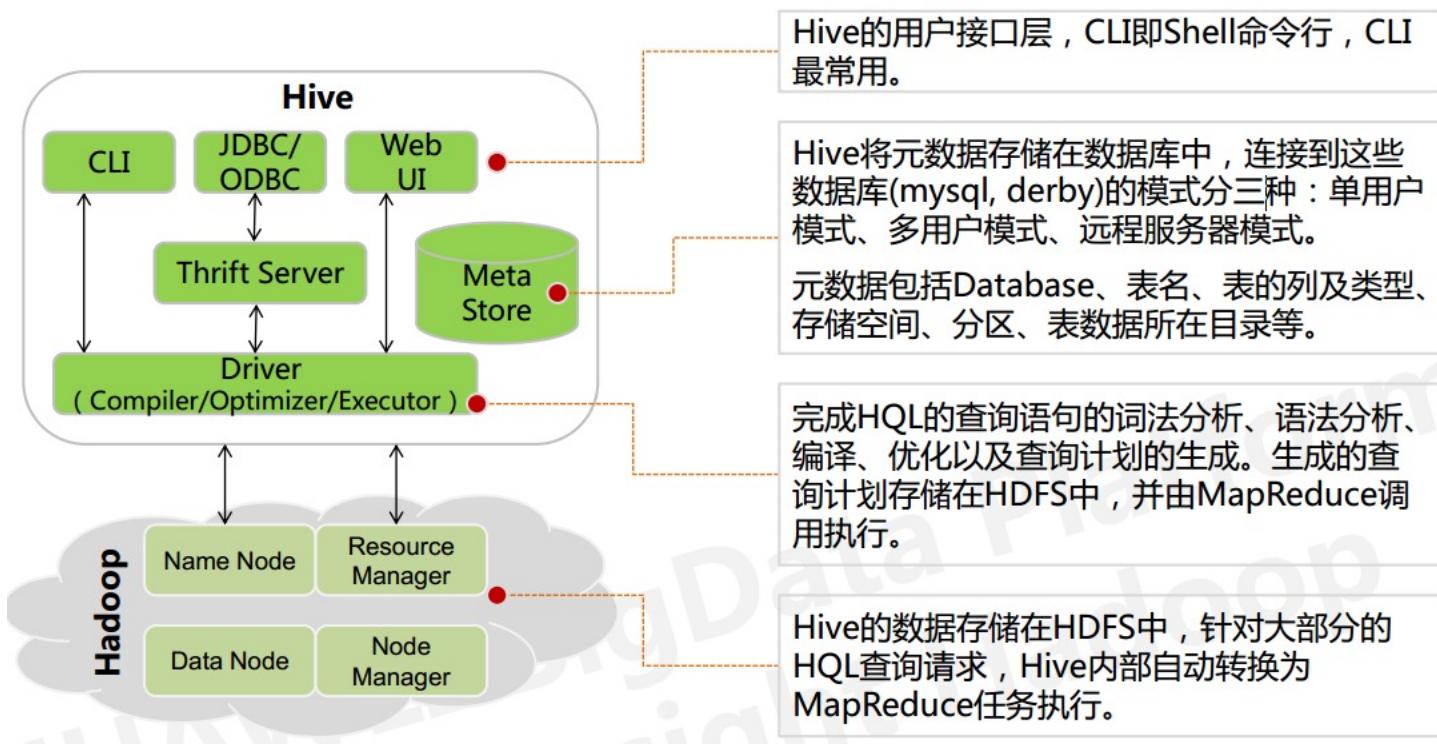
21

- **HiveQL**: 这是**Hive**的数据查询语言，与**SQL**非常类似。**Hive**提供了这个数据查询语言与用户的接口，包括一个**shell**的接口，可以进行用户的交互以及网络接口与**JDBC**接口。**JDBC**接口可以用于编程，与传统的数据库编程类似，使得程序可以直接使用**Hive**功能而无需更改。



Hive的执行流程简单示意图

22

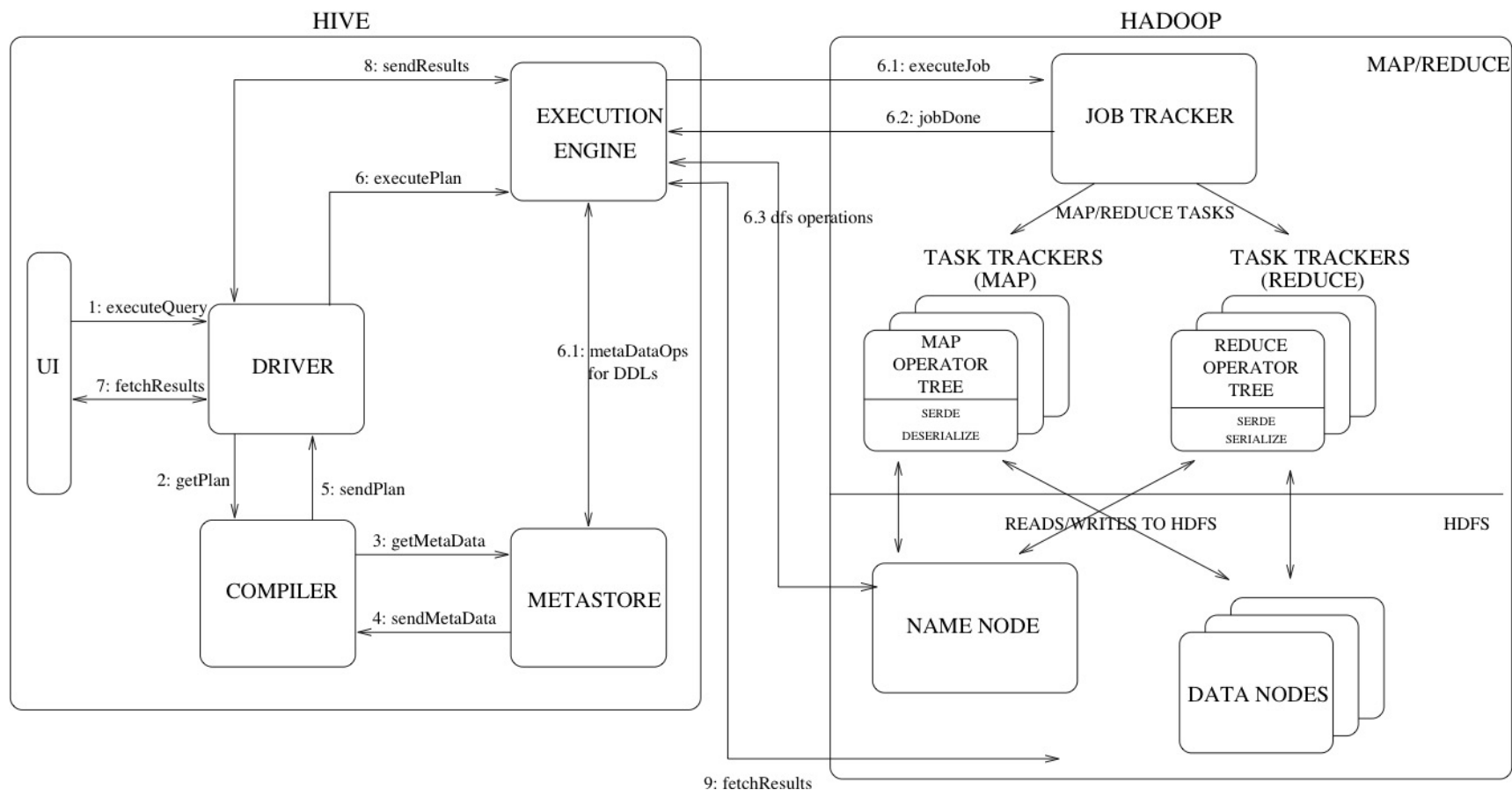


Hive 将通过CLI接入， JDBC/ODBC接入， 或者HWI接入的相关查询，通过Driver(Complier、Optimizer和Executor)，进行编译，分析优化，最后变成可执行的MapReduce。



Hive的执行流程

23



<https://cwiki.apache.org/confluence/display/Hive/Design#Design-HiveArchitecture>



Hive的数据模型

- 每一个类似于数据库的系统都首先需要定义一个数据模型，然后才是在这个数据模型之上的各种操作
- **Tables（表）**：Hive的数据模型由数据表组成。
 - ▣ 数据表中的列是有类型的（int, float, string, data, boolean）也可以是复合的类型，如list, map（类似于JSON形式的数据）
- **Partitions（分区）**：数据表可以按照一定的规则进行划分Partition。
 - ▣ 例如，通过日期的方式将数据表进行划分
- **Buckets（桶）**：数据存储的桶。
 - ▣ 在一定范围内的数据按照Hash的方式进行划分（这对于数据的抽样以及对于join的优化很有意义）



表

25

- **Hive** 表跟关系数据库里面的表类似。逻辑上，数据是存储在 **Hive** 表里面的，而表的元数据描述了数据的布局。我们可以对表执行过滤，关联，合并等操作。在 **Hadoop** 里面，物理数据一般是存储在 **HDFS** 的，而元数据是存储在关系型数据库的。
- 当我们在 **Hive** 创建表的时候，**Hive** 将以默认的方式管理表数据，也就是说，**Hive** 会默认把数据存储到 [/user/hive/warehouse](#) 目录里面。除了内部表，我们可以创建外部表，外部表需要指定数据的目录。



数据的物理分布情况

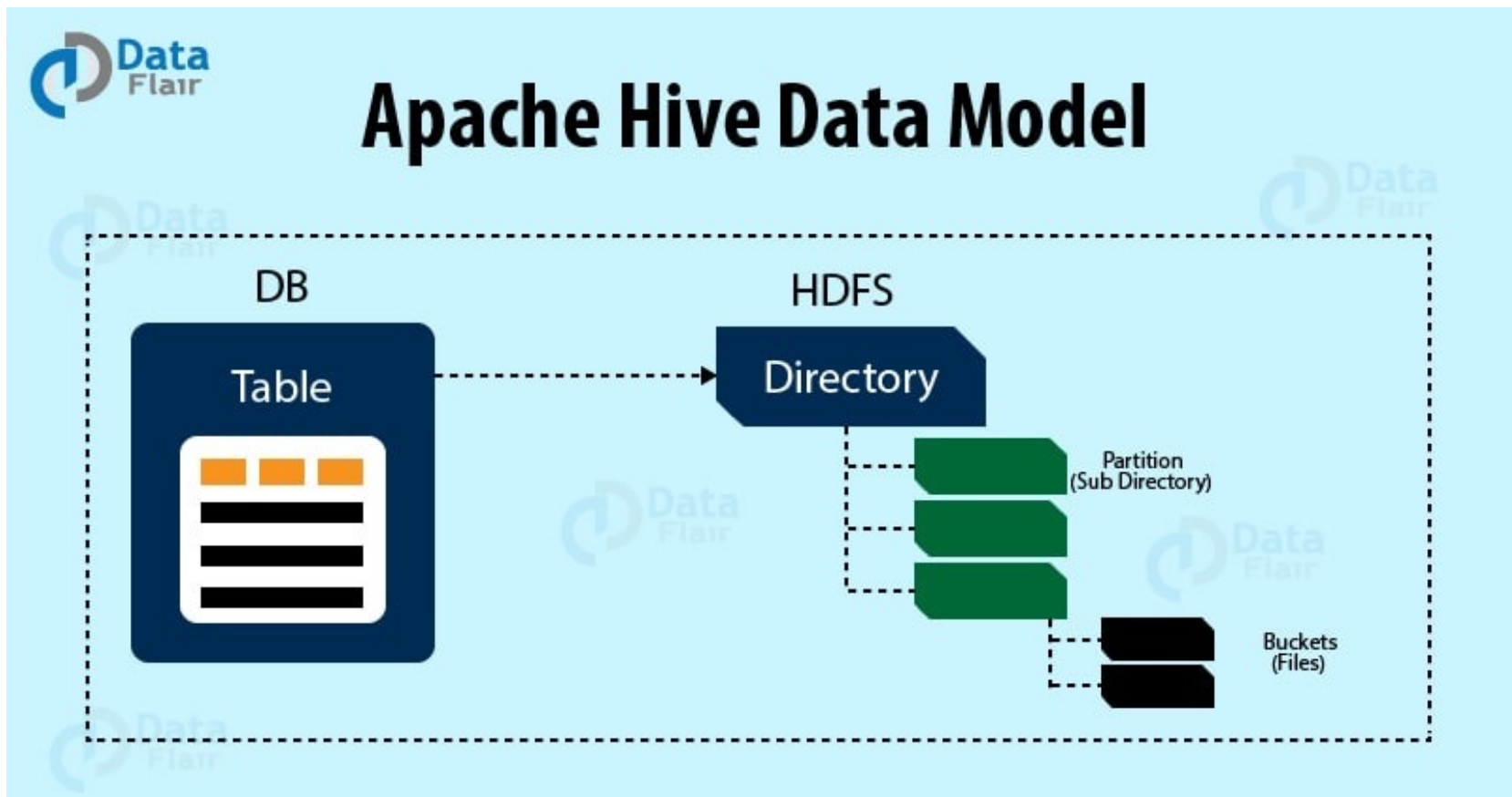
26

- **Hive**在HDFS中有固定的位置，通常被放置在HDFS的如下目录中：
[`/user/hive/warehouse`](#)
- 每个数据表被存放在**warehouse**的子目录中。进一步来说，数据划分**Partition**、数据桶**Buckets**形成了数据表的子目录
- 数据可能以任意一种形式存储，例如：
 - ▣ 使用分隔符的文本文件，或者是**SequenceFile**
 - ▣ 使用用户自定义的**SerDe**，则可以定义任意格式的文件



Hive的数据模型

27



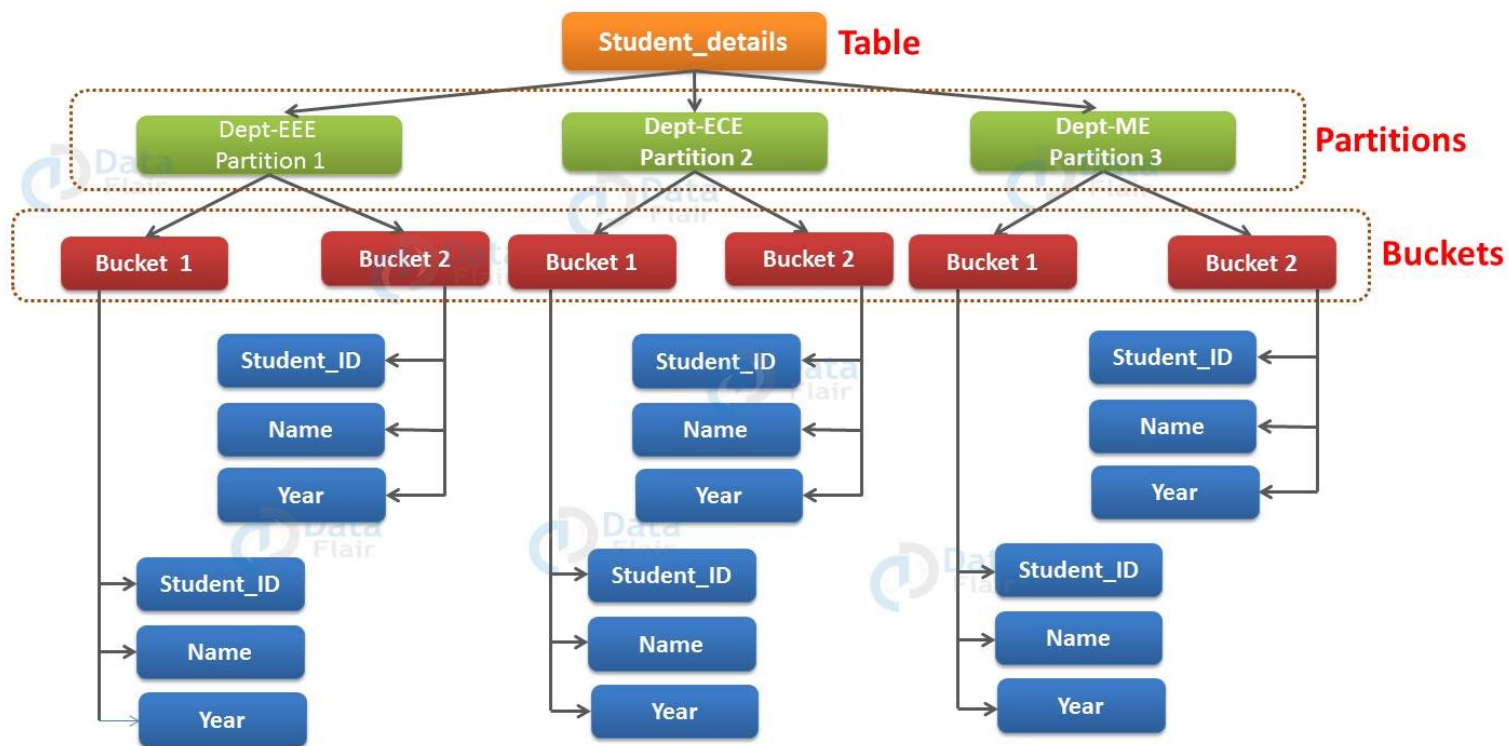


Hive的数据模型

28



Hive Data Model





分区

29

- 为了提高查询数据的效率，**Hive**提供了表分区机制。分区表基于分区键把具有相同分区键的数据存储在一个目录下，在查询某一个分区的数据的时候，只需要查询相对应目录下的数据，而不会执行全表扫描，也就是说，**Hive**在查询的时候会进行分区剪裁。每个表可以有一个或多个分区键。



桶

30

- **Hive**可以对每一个表或者是分区，进一步组织成桶，也就是说桶是更为细粒度的数据范围划分。**Hive**是针对表的某一行进行分桶。**Hive**采用对表的列值进行哈希计算，然后除以桶的个数求余的方式决定该条记录存放在哪个桶中。分桶的好处是可以获得更高的查询处理效率，使取样更高效。
- 每个桶只是表目录或者分区目录下的一个文件，如果表不是分区表，那么桶文件会存储在表目录下，如果表是分区表，那么桶文件会存储在分区目录下。所以你可以选择把分区分成 n 个桶，那么每个分区目录下就会有 n 个文件。



元数据存储：metastore

31

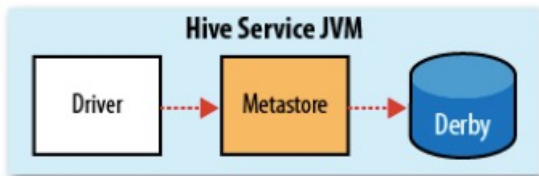
- 在**Hive**中由一系列的数据表格组成一个命名空间，关于这个命名空间的描述信息会保存在**metastore**的空间中
- 元数据使用**SQL**的形式存储在传统的关系数据库中，因此可以使用任意一种关系数据库，例如**Derby**（**Apache**的关系数据库实现），**MySQL**以及其他的多种关系数据库存储方法
- 在数据库中，保存最重要的信息是有关数据库中的数据表格描述，包括每一个表的格式定义，列的类型，物理的分布情况，数据划分情况等



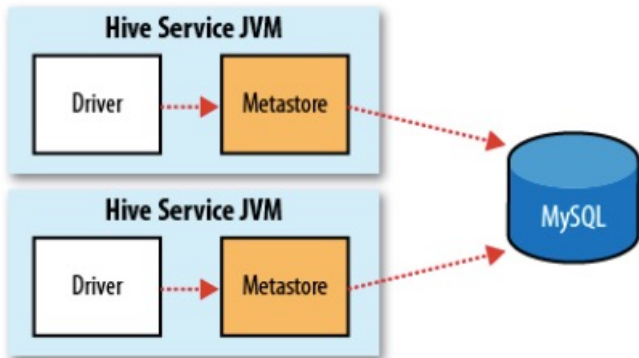
metastore的配置情况

32

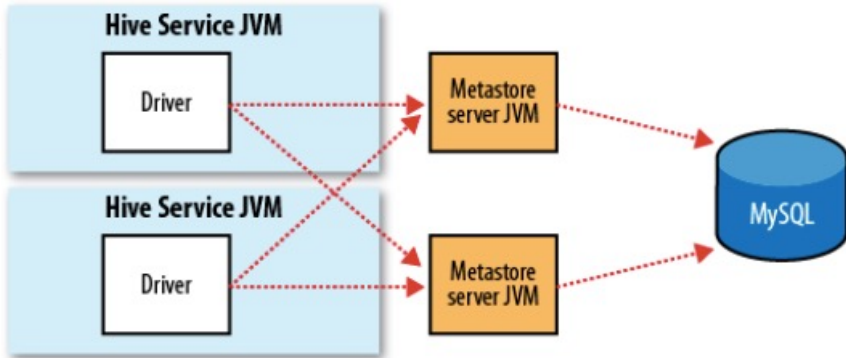
Embedded
metastore



Local
metastore



Remote
metastore



默认情况下，metastore 服务和 Hive 的服务运行在同一个 JVM 中，包含了一个内嵌的以本地磁盘作为存储的Derby（Hive 自带的数据库）数据库实例。

任何 JDBC 兼容的数据库都可以通过 `javax.jdo.option.*` 配置属性来供 metastore 使用。

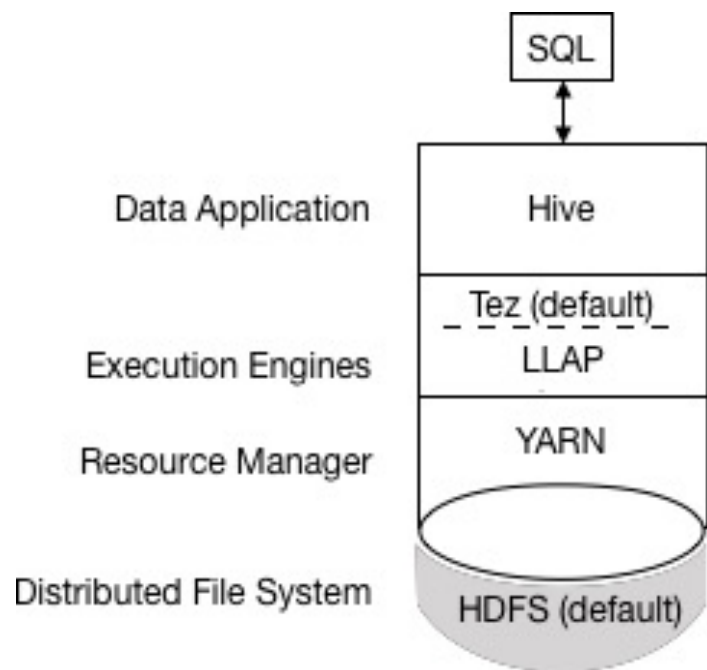
通过 `hive.metastore.uris` 设置为 metastore 服务器 URI（如果有多个服务器，可以用逗号分割），把 hive 服务设为使用远程 metastore 服务器的URI的格式为：`thrift://host:port`。



Hive 3.x变化

□ 执行引擎更改

- ▣ **Apache Tez**成为默认的**Hive**执行引擎。通过有向无环图（**DAG**）和数据传输原语的表达式，在**Tez**下执行**Hive**查询可以提高性能。



1. **Hive**编译查询。
2. **Tez**执行查询。
3. **YARN**为群集中的应用程序分配资源，并为**YARN**队列中的**Hive**作业启用授权。
4. **Hive**根据表类型更新**HDFS**或**Hive**仓库中的数据。
5. **Hive**通过**JDBC**连接返回查询结果。



Hive 3.x 变化

□ 设计影响安全性的更改

- 默认情况下，**Hadoop 3.0 Ambari**安装添加了**Apache Ranger**安全服务。**Hive**的主要授权模型是**Ranger**。此模型仅允许**Hive**访问**HDFS**。**Hive**强制执行**Ranger**中指定的访问控制。此模型提供比其他安全方案更强的安全性以及更灵活的策略管理。

□ HDFS权限更改

- **HDFS**访问控制列表（**ACL**）是**HDFS**中权限系统的扩展。**Hadoop 3.0**默认打开**HDFS**中的**ACL**，在为多个组和用户提供特定权限时，可以提高灵活性方便地将权限应用于目录树而不是单个文件。



Hive 3.x 变化

- 交易处理变更 **ACID v2**，对很多事务的特性进行了优化升级，使之更接近于关系型数据库。
 - ▣ 使用**ACID**语义修改现有**Hive**表数据，包括insert, update, delete, merge
 - ▣ 支持数据库四大特性**ACID**
 - ▣ 允许在使用长时间运行的分析查询同时进行并发更新
 - ▣ 使用**MVCC(Multi-Version concurrency Control)**架构



Hive 3.x 变化

□ 客户端

- ▣ **Hive 3**支持瘦客户端**Beeline**，用于从命令行运行查询和**Hive**管理命令。
Beeline使用与**HiveServer**的**JDBC**连接来执行所有命令。
- ▣ **Hive**客户端的更改要求使用**grunt**命令行来使用**Apache Pig**。
- ▣ **HiveServer**现在使用远程而不是嵌入式**Metastore**。

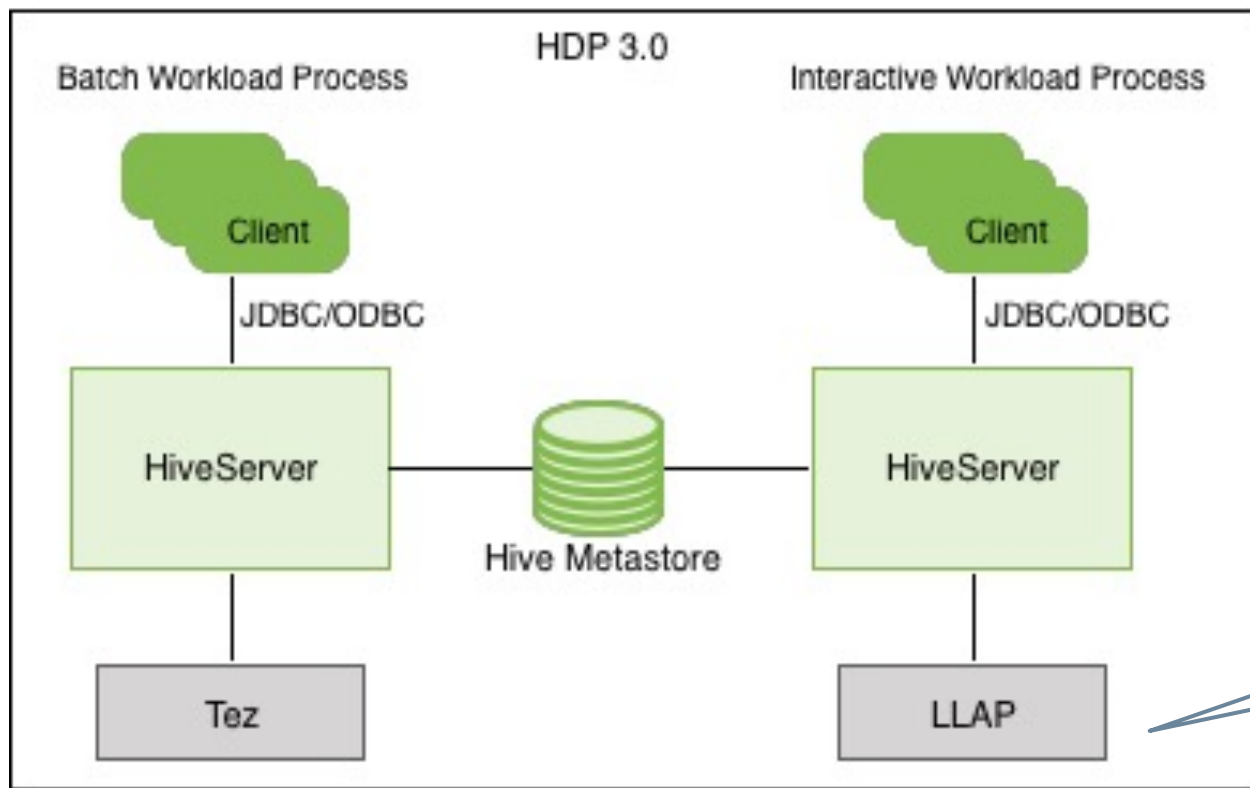
□ Spark 目录更改

- ▣ **Spark**和**Hive**现在使用独立的目录来访问相同或不同平台上的**Spark SQL**或**Hive**表。**Spark**创建的表驻留在**Spark**目录中。**Hive**创建的表位于**Hive**目录中。虽然是独立的，但这些表互操作。
- ▣ 可以使用**HiveWarehouseConnector**从**Spark**访问**ACID**和外部表。



Hive 3.x 变化

- 批处理和交互式工作负载的Hadoop 3.0查询执行体系结构



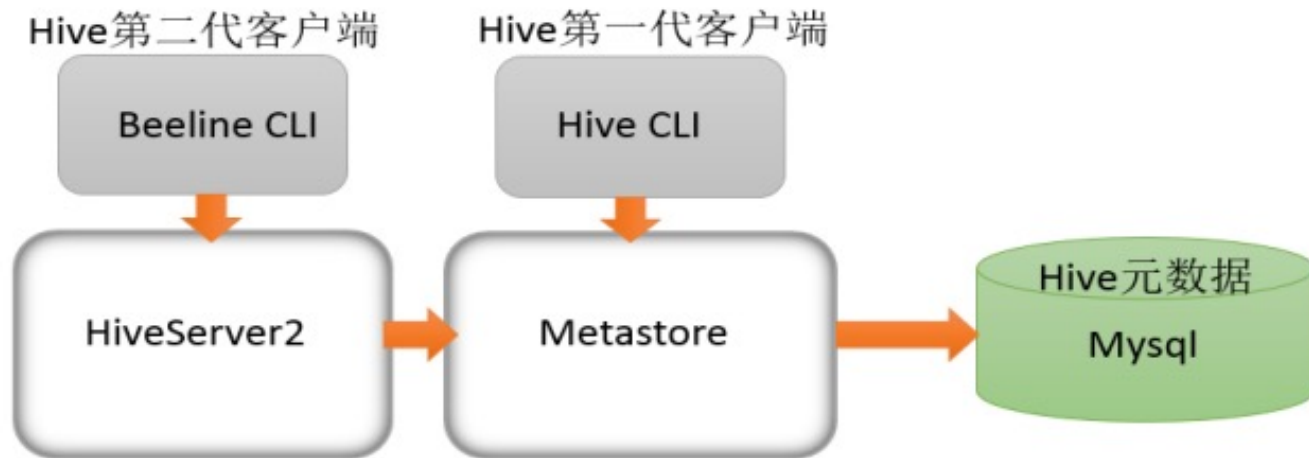
Low Latency
Analytical Processing



Hive 4.x 变化

- Hive 3.x 仍然支持 Hive CLI，但从这个版本开始，Hive CLI 已经被标记为不推荐使用（**deprecated**）。官方建议用户逐步迁移至 Beeline，尤其是面向生产环境时。
- Hive 4.x 不再支持传统的 Hive CLI。

Hive客户端和服务的关系





摘要

39

- Hive基本原理
- Hive基本操作



Hive的安装和配置

40

- 下载 <https://archive.apache.org/dist/hive/>
- 版本
 - ▣ 2.3.10 May 9, 2024 works with Hadoop 2.x.y
 - ▣ 3.1.3 April 8, 2022 works with Hadoop 3.x.y (推荐)
 - ▣ 4.0.1 October 2, 2024 works with Hadoop 3.3.6, Tez 0.10.4



Hive的安装和配置

41

- 前提
 - ▣ mysql及驱动（也可以用自带的derby数据库）
- 安装
 - ▣ 解压
 - ▣ 配置hive-env.sh
 - HADOOP_HOME
 - ▣ 配置hive-site.xml
 - 关系数据库的相关属性



Hive的安装和配置

- 要想Hive使用HDFS进行数据仓库的存储，使用MapReduce进行HQL语言的执行，需要进行相应的配置。

- ▣ hive-env.sh

- Set HADOOP_HOME

- ▣ hive-site.xml (hive-default.xml.template)

- 元数据仓库

```
<property>
```

```
  <name>javax.jdo.option.ConnectionURL</name>
```

```
  <value>jdbc:derby;;databaseName=metastore_db;create=true</value>
```

```
</property>
```

- 元数据文件目录

```
<property>
```

```
  <name>hive.metastore.warehouse.dir</name>
```

```
  <value>/user/hive/warehouse</value>
```

```
</property>
```

若已设置相关的系统环境变量，并使用默认配置，可以不修改配置文件（仅针对Hive 2.x和3.x）

这里使用的默认内嵌数据库derby，如果选择的是mysql数据库，要先安装mysql，并且将mysql的驱动包放入hive的lib文件夹



启动Hive的命令行界面shell

- 完成Hive系统的合适配置之后，打开任意一个命令行界面，执行下面的命令就可以启动hive的界面
- `$cd /hive/install/directory` 进入hive的安装目录
- `$bin /hive` 如果已经将hive加入到执行路径，则可以直接执行hive即可
- 执行成功的话，我们就可以看到hive的主界面hive>
- 等待用户输入查询命令

注意：在运行hive前要先确保Hadoop已经启动（start-dfs, start-yarn）和配置的关系数据库（derby或mysql）已经启动并完成初始化：

```
$HIVE_HOME/bin/schematool -dbType derby -initSchema
```



退出Hive的命令行界面shell

44

- ❑ `hive> quit;`
- ❑ `stop-yarn.sh`
- ❑ `stop-dfs.sh`



Hive QL

45

- **Hive**主要支持以下几类操作：
 - ▣ **DDL**: 数据定义语句，包括**CREATE, ALTER, SHOW, DESCRIBE, DROP**等
 - ▣ **DML**: 数据操作语句，包括**LOAD DATA, INSERT**。**Hive**的设计中没有考虑**UPDATE**操作。
 - ▣ **QUERY**: 数据查询语句，主要是**SELECT**语句。

注意: **Hive QL**的保留字大小写不敏感



创建数据表的命令

46

□ 显示所有的数据表

▣ `hive> show tables;`

□ 创建一个表，这个表包括两列，分别是整数类型以及字符串类型，使用文本文件表达

▣ `hive> CREATE TABLE pokes (foo INT, bar STRING);`

▣ `hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);`

▣ `hive> CREATE TABLE Shakespeare (freq int, word string)`

`row format delimited fields terminated by '\t'`

`stored as textfile;` //定义这张表使用的数据文件格式，这里指定为txt类型

如果数据是csv文件，需要忽略表头的话，可以在末尾加上这句：

`TBLPROPERTIES ('skip.header.line.count'='1');` //定义这张表的属性



描述数据表的命令

47

- 显示所创建的数据表的描述，即创建时候对于数据表的定义
 - ▣ `hive> DESCRIBE invites;`
- 修改表的语句
 - ▣ `hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);`
 - ▣ `hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');`
 - ▣ `hive> ALTER TABLE invites REPLACE COLUMNS (foo INT, bar STRING, baz INT COMMENT 'baz replaces new_col2');`



装入数据

48

□ 数据装入到Hive中

▣ `hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE
INTO TABLE pokes;`

□ NOTES

- ▣ If the '**OVERWRITE**' keyword is omitted, data files are appended to existing data sets.
- ▣ NO verification of data against the schema is performed by the load command.
- ▣ If the file is in hdfs, it is moved into the Hive-controlled file system namespace.



装入数据

49

□ 分区

- ▣ hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');
- ▣ hive> LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-08');

□ 从HDFS装入数据

- ▣ hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');



SELECTS and FILTERS

50

- ❑ `hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';`
- ❑ `hive> SELECT a.foo FROM invites a WHERE a.ds='2008-08-15' sort by a.foo asc limit 10;`
- ❑ `hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out'`
 `SELECT a.* FROM invites a WHERE a.ds='2008-08-15';`
- ❑ `hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM`
 `pokes a;`



Group By

51

- **Group by**通常和聚合函数一起使用，按照一个或者多个列进行分组，然后对每个组进行聚合操作。
- **hive> INSERT OVERWRITE TABLE events**
SELECT a.bar, count(*)
FROM invites a
WHERE a.foo > 0 GROUP BY a.bar;



Group By

52

聚合函数

返回类型	签名	描述
BIGINT	count(*), count(expr),	count(*) - 返回检索行的总数。
DOUBLE	sum(col), sum(DISTINCT col)	返回该组或该组中的列的不同值的分组和所有元素的总和。
DOUBLE	avg(col), avg(DISTINCT col)	返回上述组或该组中的列的不同值的元素的平均值。
DOUBLE	min(col)	返回该组中的列的最小值。
DOUBLE	max(col)	返回该组中的列的最大值。



Join

53

- Join主要对两个表通过两个相同的字段进行连接，并查询相关的结果。
- `hive> SELECT t1.bar, t1.foo, t2.foo
FROM pokes t1 JOIN invites t2 ON
t1.bar = t2.bar;`



排序

54

□ order by

- 全局排序，只有一个reducer
- ASC: 升序（默认）； DESC: 降序
- `hive> select * from emp order by sal desc;` //查询员工信息按工资降序排列
- 在严格模式下（`set hive.mapred.mode = strict`），`order by`必须要和 `limit` 一起用，否则会报错的。Hive 一般默认使用的是非严格模式（`set hive.mapred.mode=nonstrict`）

□ sort by

- `sort by`为每个reducer产生一个排序文件，首先要设置reducer个数(>1)。 `sort by`只会保证每个reducer的输出有序，并不保证全局有序。
- `hive> set mapreduce.job.reduces=3;`
- `hive> select * from emp sort by deptno desc;` //根据部门编号降序查看员工信息



排序

55

□ distribute by

- 在有些情况下，我们需要控制某个特定行应该到哪个reducer，通常是为了进行后续的聚集操作。**distribute by** 子句可以做这件事。**distribute by**类似MR中**partition**（自定义分区），进行分区，结合**sort by**使用。**distribute by**的分区规则是根据分区字段的 **hash** 码与 **reduce** 的个数进行模除后，余数相同的分到一个区。
- **DISTRIBUTE BY** 语句要写在 **SORT BY** 语句之前。
- `hive> set mapreduce.job.reduces=3;`
- `hive> insert overwrite local directory '/opt/module/data/distribute-result' select * from emp
distribute by deptno sort by empno desc; //先按照部门编号分区，再按照员工编号降序排序。`



排序

56

□ cluster by

- 当**distributed by**和**sort by**字段相同时，可以使用**cluster by**方式。
- **cluster by**除了具有**distributed by**的功能外还兼具**sort by**的功能。但是排序只能是升序排序，不能指定排序规则为**ASC**或者**DESC**。
- `hive> select * from emp cluster by deptno;` //先按照部门编号分区并按升序排序
- 等同于
- `hive> select * from emp distributed by deptno sort by deptno;`



Hive: Example

- Hive looks similar to an SQL database
- Relational join on two tables:
 - ▣ Table of word counts from Shakespeare collection
 - ▣ Table of word counts from the bible

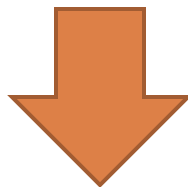
```
SELECT s.word, s.freq, k.freq FROM shakespeare s
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
ORDER BY s.freq DESC LIMIT 10;
```

the	25848	62394
I	23031	8854
and	19671	38985
to	18038	13526
of	16700	34654
a	14170	8057
you	12702	2720
my	11297	4135
in	10797	12445
is	8882	6884



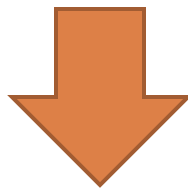
Hive: Behind the Scenes

```
SELECT s.word, s.freq, k.freq FROM shakespeare s  
JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1  
ORDER BY s.freq DESC LIMIT 10;
```



(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s)  
word) (. (TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT  
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.  
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k)  
freq) 1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```



(one or more of MapReduce jobs)



Hive: Behind the Scenes

STAGE DEPENDENCIES:

Stage-1 is a root stage
Stage-2 depends on stages: Stage-1
Stage-0 is a root stage

STAGE PLANS:

Stage: Stage-1
Map Reduce
Alias -> Map Operator Tree:

s
TableScan
alias: s
Filter Operator
predicate:
 expr: (freq >= 1)
 type: boolean
Reduce Output Operator
key expressions:
 expr: word
 type: string
sort order: +
Map-reduce partition

columns:
 expr: word
 type: string
tag: 0
value expressions:
 expr: freq
 type: int
 expr: word
 type: string

k
TableScan
alias: k
Filter Operator
predicate:
 expr: (freq >= 1)
 type: boolean
Reduce Output Operator
key expressions:
 expr: word
 type: string
sort order: +
Map-reduce partition
columns:
 expr: word
 type: string
tag: 1
value expressions:
 expr: freq
 type: int

Reduce Operator Tree:

Join Operator
condition map:
 Inner Join 0 to 1
condition expressions:
 0 {VALUE._col0} {VALUE._col1}
 1 {VALUE._col0}
outputColumnNames: _col0, _col1, _col2
Filter Operator
predicate:
 expr: ((_col0 >= 1) and (_col2 >= 1))
 type: boolean
Select Operator
expressions:
 expr: _col1
 type: string
 expr: _col0
 type: int
 expr: _col2
 type: int
outputColumnNames: _col0, _col1, _col2
File Output Operator
compressed: false
GlobalTableId: 0
table:
 input format:
 org.apache.hadoop.mapred.SequenceFileInputFormat
 output format:
 org.apache.hadoop.hive ql.io.HiveSequenceFileOutputFormat

Stage: Stage-2

Map Reduce
Alias -> Map Operator Tree:
hdfs://localhost:8022/tmp/hive-training/364214370/10002

Reduce Output Operator
key expressions:
 expr: _col1
 type: int
sort order: -
tag: -1

value expressions:
 expr: _col0
 type: string
 expr: _col1
 type: int
 expr: _col2
 type: int

Reduce Operator Tree:

Extract
Limit
File Output Operator
compressed: false
GlobalTableId: 0
table:
 input format:
 org.apache.hadoop.mapred.TextInputFormat
 output format:
 org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat

Stage: Stage-0
Fetch Operator
limit: 10



Hive: Example

60

□ 数据文件

- 1 Rod 18 study-game-driver std_addr:Beijing-work_addr:shanghai
- 2 Tom 21 study-game-driver std_addr:Beijing-work_addr:Beijing
- 3 Jerry 33 study-game-football std_addr:Beijing-work_addr:Shenzhen
- 4 Bob 23 study-game-music std_addr:Beijing-work_addr:Nanjing

□ 创建Hive表

- ▣ hive> CREATE TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)
COMMENT 'This is the person table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY '-'
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;

□ 导入数据

- ▣ hive> LOAD DATA LOCAL INPATH person.txt OVERWRITE INTO TABLE person;



表的分区

61

- 可以把数据依照单个或多个列进行分区，通常按照时间、地域进行分区。为了达到性能表现的一致性，对不同列的划分应该让数据尽可能均匀分布。分区应当在建表时设置。
 - ▣ `hive> CREATE TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)`
`COMMENT 'This is the person table'`
`PARTITIONED BY (dt STRING)`
`ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'`
`COLLECTION ITEMS TERMINATED BY '-'`
`MAP KEYS TERMINATED BY ':'`
`STORED AS TEXTFILE;`
 - ▣ `hive> LOAD DATA LOCAL INPATH 'person.txt' OVERWRITE INTO TABLE person partition (dt='2018-03-15');`
 - ▣ `hive> SELECT addr['work_addr'] from person WHERE dt='2018-03-15';`



表的分区

62

- 分区表实际上就是对应一个 **HDFS** 文件系统上的独立的文件夹，该文件夹下是该分区所有的数据文件。**Hive**中的分区就是分目录，把一个大的数据集根据业务需要分割成小的数据集。在查询时通过 **WHERE** 子句中的表达式选择查询所需要的指定的分区，这样的查询效率会提高很多。

- `hive> ALTER TABLE person add partition (dt='2018-03-16');` //增加分区

- `hive> ALTER TABLE person drop partition (dt='2018-03-16');` //删除分区

- `hive> describe formatted person;` //查看分区表结构

- `hive> show partitions person;` //查看多少个分区

- 二级分区

- `hive> create table dept_partition2 (deptno int, dname string, loc string)`

- `partitioned by (day string, hour string) row format delimited fields terminated by '\t';`

- `Hive> select * from dept_partition2 where day='20220225' and hour='12';`



表的分区

63

- 关系型数据库中，对分区表 **Insert** 数据时候，数据库自动会根据分区字段的值，将数据插入到相应的分区中，**Hive** 中也提供了类似的机制，即动态分区 (**Dynamic Partition**)，只不过，使用 **Hive** 的动态分区，需要进行相应的配置。
 - ▣ 设置为非严格模式（动态分区的模式，默认 **strict**，表示必须指定至少一个分区为 静态分区，**nonstrict** 模式表示允许所有的分区字段都可以使用动态分区。）
 - ▣ `set hive.exec.dynamic.partition=true;`
 - ▣ `set hive.exec.dynamic.partition.mode=nonstrict;`
 - ▣ 例如：：将 **dept** 表中的数据按照地区（**loc** 字段），插入到目标表 **dept_partition** 的相应 分区中。
 - ▣ `hive> create table dept_par4(id int, name string) partitioned by (loc int) row format delimited fields terminated by '\t';`
 - ▣ 设置动态分区
 - ▣ `hive> insert into table dept_par4 partition(loc) select deptno, dname, loc from dept;`
 - ▣ 注：查询语句的最后字段默认为分区字段。



表的分桶

64

- 分桶是相对于分区进行更细粒度的划分。在分区数量过于庞大以至于可能导致文件系统崩溃时，就需要使用分桶来解决问题。
- 分桶将整个数据内容按照某列属性值的hash值进行区分。分桶同样应当在建表时就建立。
 - ▣ `hive> set hive.strict.checks.bucketing=false;`
 - ▣ `hive> CREATE TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)
PARTITIONED BY (dt STRING)
CLUSTERED BY (id) into 3 buckets
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY '-'
MAP KEYS TERMINATED BY ':'
STORED AS TEXTFILE;`



表的分桶

65

□ 导入数据

▣ `hive> LOAD DATA LOCAL INPATH 'person.txt' OVERWRITE INTO TABLE person partition (dt='2018-03-15');`

□ 采样

▣ `hive> SELECT * from person TABLESAMPLE (BUCKET 1 OUT OF 3);`

- `tablesample(bucket X out of Y)` 并不是仅仅提取一个桶的数据，而是对所有桶的数据进行均匀分布的采样，按比例提取出 X/Y 的数据。它不受桶的物理限制，而是基于随机选择。



内部表和外部表

66

- 内部表：在创建的时候会把数据移动到数据仓库所指向的位置；在删除的时候会将元数据和数据一起删除。如果仅仅是**Hive**内部使用，可以使用内部表。
 - ▣ `hive > CREATE TABLE managed_table (dummy STRING);`
 - ▣ `hive > LOAD DATA INPATH '/user/tom/data.txt' INTO table managed_table;`
- 根据上面的代码，**Hive**会把文件**data.txt**文件存储在**managed_table**表的**warehouse**目录下，即hdfs://user/hive/warehouse/managed_table目录。如果我们用**drop**命令把表删除，将会把表以及表里面的数据和表的元数据都一起删除。
 - ▣ `hive > DROP TABLE managed_table;`



内部表和外部表

67

- 外部表：仅仅记录数据所在的位置；在删除的时候仅仅删除元数据，真正的数据不会删除。
 - ▣ 创建空表（通过 **external** 关键词表明创建的表是外部表），然后导入数据；
 - ▣ 创建表的时候指定数据文件的位置，用 **external** 关键词创建外部表，使用 **location** 关键词指定数据文件的位置；利用 **EXTERNAL** 关键字创建外部表，**Hive** 不会去管理表数据，所以它不会把数据移到 **/user/hive/warehouse** 目录下。甚至在执行创建语句的时候，它不会去检查建表语句中指定的外部数据路径是否存在。可以在表创建之后，再创建数据。
 - ▣ 删除外部表的时候，**Hive** 只有删除表的元数据，而不会删除表数据。



外部表

68

- hive> CREATE **external** TABLE person (id INT, name STRING, age INT, fav ARRAY<STRING>, addr MAP<STRING, STRING>)
COMMENT 'This is the person table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
COLLECTION ITEMS TERMINATED BY '-'
MAP KEYS TERMINATED BY ':'
LOCATION 'hdfs://localhost:9000/root/ '
STORED AS TEXTFILE;
- hive> dfs -ls /user/hive/warehouse/
- hive> dfs -ls /user/hive/warehouse/person



自定义函数

69

□ User Defined Function (UDF)

- ▣ UDF: 作用于单条数据, 并且输出一个数据行, 如字符串函数、日期函数。
- ▣ UDAF: 可接受多个数据输入, 同时输出一个数据行, 如COUNT、MIN、MAX等聚集函数。
- ▣ UDTF: 作用于单个数据行, 同时产生多个输出数据行。



Hive 总结

70

- **Hive**提供了一种类似于**SQL**的查询语言，使得其能够用于用户的交互查询
- 与传统的数据库类似，**Hive**提供了多个数据表之间的联合查询，能够完成高效的多个数据表之间的查询
- 通过底层执行引擎的工作，**Hive**将**SQL**语言扩展到很大的查询规模

THANK YOU



南京大學
NANJING UNIVERSITY

南京大学计算机软件研究所
Institute of Computer Software, Nanjing University