HIVE 快速入门简明教程

版 本	V 0.1
作 者	许向可
时 间	2018.5

一、HIVE 介绍

1. What is Hive

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 QL，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。

2. HIVE 架构

Hive 的结构可以分为以下几部分：

- 用户接口：包括 CLI, Client, WUI
- 元数据存储。通常是存储在关系数据库如 mysql, derby 中
- 解释器、编译器、优化器、执行器
- Hadoop：用 HDFS 进行存储，利用 MapReduce 进行计算

1、 用户接口主要有三个：CLI，Client 和 WUI。其中最常用的是 CLI，Cli 启动的时候，会同时启动一个 Hive 副本。Client 是 Hive 的客户端，用户连接至 Hive Server。在启动 Client 模式的时候，需要指出 Hive Server 所在节点，并且在该节点启动 Hive Server。 WUI 是通过浏览器访问 Hive。

2、 Hive 将元数据存储在数据库中，如 mysql、derby。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

3、 解释器、编译器、优化器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在 HDFS 中，并在随后有 MapReduce 调用执行。

4、Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 * 的查询，比如 select * from tbl 不会生成 MapRedcue 任务）。

3. Hive 和 Hadoop 关系

Hive 构建在 Hadoop 之上，

- HQL 中对查询语句的解释、优化、生成查询计划是由 Hive 完成的
- 所有的数据都是存储在 Hadoop 中
- 查询计划被转化为 MapReduce 任务，在 Hadoop 中执行（有些查询没有 MR 任务，如：select * from table）
- Hadoop 和 Hive 都是用 UTF-8 编码的

4. Hive 和普通关系数据库的异同

	Hive	RDBMS
查询语言	HQL	SQL
数据存储	HDFS	Raw Device or Local FS

索引	无	有
执行	MapReduce	Excutor
执行延迟	高	低
处理数据规模	大	小

1. 查询语言。由于 SQL 被广泛的应用在数据仓库中，因此，专门针对 Hive 的特性设计了类 SQL 的查询语言 HQL。熟悉 SQL 开发的开发者可以很方便的使用 Hive 进行开发。

2. 数据存储位置。Hive 是建立在 Hadoop 之上的，所有 Hive 的数据都是存储在 HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。

3. 数据格式。Hive 中没有定义专门的数据格式，数据格式可以由用户指定，用户定义数据格式需要指定三个属性：列分隔符（通常为空格、”\t”、”\x001″）、行分隔符（”\n”）以及读取文件数据的方法（Hive 中默认有三个文件格式 TextFile，SequenceFile 以及 RCFile）。由于在加载数据的过程中，不需要从用户数据格式到 Hive 定义的数据格式的转换，因此，Hive 在加载的过程中不会对数据本身进行任何修改，而只是将数据内容复制或者移动到相应的 HDFS
目录中。而在数据库中，不同的数据库有不同的存储引擎，定义了自己的数据格式。所有数据都会按照一定的组织存储，因此，数据库加载数据的过程会比较耗时。

4. 数据更新。由于 Hive 是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive 中不支持对数据的改写和添加，所有的数

据都是在加载的时候中确定好的。而数据库中的数据通常是需要经常进行修改的,因此可以使用 INSERT INTO … VALUES 添加数据,使用 UPDATE… SET 修改数据。

5. 索引。之前已经说过，Hive 在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些 Key 建立索引。Hive 要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入， Hive 可以并行访问数据，因此即使没有索引，对于大数据量的访问，Hive 仍然可以体现出优势。数据库中，通常会针对一个或者几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了
Hive 不适合在线数据查询。

6. 执行。Hive 中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的（类似 select * from tbl 的查询不需要 MapReduce）。而数据库通常有自己的执行引擎。

7. 执行延迟。之前提到，Hive 在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致 Hive 执行延迟高的因素是 MapReduce 框架。由于 MapReduce 本身具有较高的延迟，因此在利用 MapReduce 执行 Hive 查询时，也会有较高的延迟。相对的，数据库的执行延迟较低。当然，这个低是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive 的并行计算显然能体现出优势。

8. 可扩展性。由于 Hive 是建立在 Hadoop 之上的，因此 Hive 的可

扩展性是和 Hadoop 的可扩展性是一致的（世界上最大的 Hadoop 集群在 Yahoo!，2009 年的规模在 4000 台节点左右）。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最先进的并行数据库 Oracle 在理论上的扩展能力也只有 100 台左右。

9. 数据规模。由于 Hive 建立在集群上并可以利用 MapReduce 进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

5. HIVE 元数据库

Hive 将元数据存储在 RDBMS 中,一般常用的有 MYSQL 和 DERBY。

6. HIVE 的数据存储

首先，Hive 没有专门的数据存储格式，也没有为数据建立索引，用户可以非常自由的组织 Hive 中的表，只需要在创建表的时候告诉 Hive 数据中的列分隔符和行分隔符，Hive 就可以解析数据。

其次，Hive 中所有的数据都存储在 HDFS 中，Hive 中包含以下数据模型：Table，External Table，Partition，Bucket。

Hive 中的 Table 和数据库中的 Table 在概念上是类似的，每一个 Table 在 Hive 中都有一个相应的目录存储数据。例如，一个表 xiaojun，它在 HDFS 中的路径为：/ warehouse /xiaojun，其中，wh 是在 hive-site.xml 中由 ${tastore.warehouse.dir} 指定的数据仓库的目录，所有的 Table 数据（不包括 External Table）都保存在这个目录中。

Partition 对应于数据库中的 Partition 列的密集索引，但是 Hive

中 Partition 的组织方式和数据库中的很不相同。在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在对应的目录中。例如：xiaojun 表中包含 dt 和 city 两个 Partition，则对应于 dt = 20100801, ctry = US 的 HDFS 子目录为：/ warehouse /xiaojun/dt=20100801/ctry=US；对应于 dt = 20100801, ctry = CA 的 HDFS 子目录为；/ warehouse /xiaojun/dt=20100801/ctry=CA

Buckets 对指定列计算 hash，根据 hash 值切分数据，目的是为了并行，每一个 Bucket 对应一个文件。将 user 列分散至 32 个 bucket，首先对 user 列的值计算 hash，对应 hash 值为 0 的 HDFS 目录为：/ warehouse /xiaojun/dt =20100801/ctry=US/part-00000；hash 值为 20 的 HDFS 目录为：/ warehouse /xiaojun/dt =20100801/ctry=US/part-00020

External Table 指向已经在 HDFS 中存在的数据，可以创建 Partition。它和 Table 在元数据的组织上是相同的，而实际数据的存储则有较大的差异。

Table 的创建过程和数据加载过程（这两个过程可以在同一个语句中完成），在加载数据的过程中，实际数据会被移动到数据仓库目录中；之后对数据对访问将会直接在数据仓库目录中完成。删除表时，表中的数据和元数据将会被同时删除。

External Table 只有一个过程，加载数据和创建表同时完成（CREATE EXTERNAL TABLE ……LOCATION），实际数据是存储在 LOCATION 后面指定的 HDFS 路径中，并不会移动到数据仓库目录中。当删除一个 External

Table 时，仅删除

二、HIVE 基本操作

1. 创建数据库

创建数据库是用来创建数据库在 Hive 中语句。在 Hive 数据库是一个命名空间或表的集合。此语法声明如下：

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

在这里，IF NOT EXISTS 是一个可选子句，通知用户已经存在相同名称的数据库。可以使用 SCHEMA 在 DATABASE 的这个命令。下面的查询执行创建一个名为 userdb 数据库：

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

或

```
hive> CREATE SCHEMA userdb;
```

下面的查询用于验证数据库列表：

```
hive> SHOW DATABASES;

default

userdb
```

2. 删除数据库

DROP DATABASE 是删除所有的表并删除数据库的语句。它的语法如下：

```
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database_name
[RESTRICT|CASCADE];
```

下面的查询用于删除数据库。假设要删除的数据库名称为 userdb。

```
hive> DROP DATABASE IF EXISTS userdb;
```

以下是使用 CASCADE 查询删除数据库。这意味着要全部删除相应的表在删除数据库之前。

```
hive> DROP DATABASE IF EXISTS userdb CASCADE;
```

以下使用 SCHEMA 查询删除数据库。

```
hive> DROP SCHEMA userdb;
```

3. 创建表

Create Table 是用于在 Hive 中创建表的语句。语法和示例如下：

语法

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [COMMENT col_comment], ...)][COMMENT table_comment][ROW
FORMAT row_format][STORED AS file_format]
```

示例

3.1 创建内部表

1、假设需要使用 CREATE TABLE 语句创建一个名为 employee 表。下表列出了 employee 表中的字段和数据类型：

Sr.No	字段名称	数据类型
1	Eid	int

2	Name	String
3	Salary	Float
4	Designation	string

2、下面的数据是一个注释，行格式字段，如字段终止符，行终止符，并保存的文件类型。

```
COMMENT  'Employee details'
FIELDS TERMINATED BY  '\t'
LINES TERMINATED BY  '\n'
STORED IN TEXT FILE
```

3、employee 建表语句如下：

```
CREATE TABLE IF NOT EXISTS userdb.employee
(eid Int comment "num",
 name String,
 salary String,
 destination String
 )
 comment "Employee details"
 ROW FORMAT
  DELIMITED FIELDS TERMINATED BY '\t'
  lines Terminated by "\n"
  STORED AS TEXTFILE
  location "/hive";
```

3.2 创建外部表

1、假设我们在 Hdfs 上的/data/hive_test 目录下有如下数据：

```
Zhangsan|shuxue|89
Wangwu|shuxue|39
Lisi|shuxue|70
Mayun|shuxue|0
Zhangsan|yuwen|79
```

```
Wangwu|yuwen|90
Lisi|yuwen|98
Zhangsan|yingyu|40
Wangwu|yingyu|70
Lisi|yingyu|40
Mayun|yingyu|100
```

2、字段说明

Sr.No	字段名称	数据类型
1	name	string
2	course	String
3	achievement	Double

3、根据以上信息来创建 Hive 外部表，建表语句如下：

```
create external table achievement
(name string,
course string,
achievement Double)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '|'
  LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION '/data/hive_test';
```

3.3 Hive 内外部表的区别：

1、在导入数据到外部表，数据并没有移动到自己的数据仓库目录下，也就是说外部表中的数据并不是由它自己来管理的，而表则不一样；

2、在删除表的时候，Hive 将会把属于表的元数据和数据全部删掉；而删除外部表的时候，Hive 仅仅删除外部表的元数据，数据是不会删除的！

4. 修改表

它是在 Hive 中用来修改的表。

语法:

ALTER TABLE name RENAME TO new_name

ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])

ALTER TABLE name CHANGE column_name new_name new_type

ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

Rename To… 语句

下面是查询重命名表，把 employee 修改为 emp。

hive> ALTER TABLE employee RENAME TO emp;

5. 删除表

语法如下：

```
DROP TABLE [IF EXISTS] table_name;
```

以下查询删除一个名为 employee 的表:

```
hive> DROP TABLE IF EXISTS employee;
```

对于成功执行查询，能看到以下回应：

```
OKTime taken: 5.3 seconds
hive>
```

6. 分区

Hive 组织表到分区。它是将一个表到基于分区列，如日期，城市和部门的值相关方式。使用分区，很容易对数据进行部分查询。

表或分区是细分成桶，以提供额外的结构，可以使用更高效的查询的数据。桶的工作是基于表的一些列的散列函数值。

例如，我们这里有 2012 和 2013 年的成绩单分别存放在 HDFS/data/hive_test/2012 和 /data/hive_test/2013 目录，我们通过年份分区。下面的示例演示如何分区的文件和数据：

2012 年成绩单：

```
Zhangsan|shuxue|89
Wangwu|shuxue|39
Lisi|shuxue|70
Mayun|shuxue|0
Zhangsan|yuwen|79
Wangwu|yuwen|90
Lisi|yuwen|98
Zhangsan|yingyu|40
Wangwu|yingyu|70
Lisi|yingyu|40
Mayun|yingyu|100
```

2013 年成绩单：

```
Zhangsan|shuxue|69
Wangwu|shuxue|30
Lisi|shuxue|80
Mayun|shuxue|10
Zhangsan|yuwen|89
Wangwu|yuwen|99
Lisi|yuwen|97
```

```
Zhangsan|yingyu|60

Wangwu|yingyu|80

Lisi|yingyu|50

Mayun|yingyu|100
```

我们先来创建一个以 year 为分区的分区表 archiv

```
create external table achiev
(name string,
course string,
achievement Double)
partitioned by (year string)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '|'
  LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION '/data/hive_test';
```

添加分区语法：

ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec[LOCATION 'location1'] partition_spec [LOCATION 'location2'] ...;

partition_spec:: (p_column = p_col_value, p_column = p_col_value, ...)

以下是添加分区语句：

```
Alter table achiev add partition(year="2012") location "/data/hive_test/2012"
partition(year="2013") location "/data/hive_test/2013";
```

三、HIVE 运算

1. 关系运算

这些操作符被用来比较两个操作数。下表描述了在 Hive 中可用的关系运算符：

运算符	操作	描述
A = B	所有基本类型	如果表达 A 等于表达 B，结果 TRUE ，否则 FALSE。
A != B	所有基本类型	如果 A 不等于表达式 B 表达返回 TRUE ，否则 FALSE。
A < B	所有基本类型	TRUE，如果表达式 A 小于表达式 B，否则 FALSE。
A <= B	所有基本类型	TRUE，如果表达式 A 小于或等于表达式 B，否则 FALSE。
A > B	所有基本类型	TRUE，如果表达式 A 大于表达式 B，否则 FALSE。
A >= B	所有基本类型	TRUE，如果表达式 A 大于或等于表达式 B，否则 FALSE。
A IS NULL	所有类型	TRUE，如果表达式的计算结果为 NULL，否则 FALSE。
A IS NOT NULL	所有类型	FALSE，如果表达式 A 的计算结果为 NULL，否则 TRUE。
A LIKE B	字符串	TRUE，如果字符串模式 A 匹配到 B，否则 FALSE。
A RLIKE B	字符串	NULL，如果 A 或 B 为 NULL；TRUE，如果 A 任何子字符串匹配 Java 正则表达式 B；否则 FALSE。
A REGEXP B	字符串	等同于 RLIKE.

2. 算术运算

这些运算符支持的操作数各种常见的算术运算。所有这些返回数字类型。下表描述了在 Hive 中可用的算术运算符：

运算符	操作	描述
A + B	所有数字类型	A 加 B 的结果
A - B	所有数字类型	A 减去 B 的结果
A * B	所有数字类型	A 乘以 B 的结果
A / B	所有数字类型	A 除以 B 的结果
A % B	所有数字类型	A 除以 B.产生的余数
A & B	所有数字类型	A 和 B 的按位与结果
A | B	所有数字类型	A 和 B 的按位或结果
A ^ B	所有数字类型	A 和 B 的按位异或结果
~A	所有数字类型	A 按位非的结果

3. 逻辑运算

运算符是逻辑表达式。所有这些返回 TRUE 或 FALSE。

运算符	操作	描述
A AND B	boolean	TRUE，如果 A 和 B 都是 TRUE，否则 FALSE。
A && B	boolean	类似于 A AND B.
A OR B	boolean	TRUE，如果 A 或 B 或两者都是 TRUE，否则 FALSE。
A || B	boolean	类似于 A OR B.
NOT A	boolean	TRUE，如果 A 是 FALSE，否则 FALSE。
!A	boolean	类似于 NOT A.

四、HIVE 内置函数

Hive 支持以下内置函数：

返回类型	签名	描述
BIGINT	round(double a)	返回 BIGINT 最近的 double 值。
BIGINT	floor(double a)	返回最大 BIGINT 值等于或小于 double。
BIGINT	ceil(double a)	它返回最小 BIGINT 值等于或大于 double。
double	rand(), rand(int seed)	它返回一个随机数，从行改变到行。
string	concat(string A, string B,...)	它返回从 A 后串联 B 产生的字符串
string	substr(string A, int start)	它返回一个起始，从起始位置的子字符串，直到 A.结束
string	substr(string A, int start, int length)	返回从给定长度的起始 start 位置开始的字符串。
string	upper(string A)	它返回从转换的所有字符为大写产生的字符串。
string	ucase(string A)	和上面的一样
string	lower(string A)	它返回转换 B 的所有字符为小写产生的字符串。
string	lcase(string A)	和上面的一样
string	trim(string A)	它返回字符串从 A.两端修剪空格的结果
string	ltrim(string A)	它返回 A 从一开始修整空格产生的字符串(左手侧)
string	rtrim(string A)	rtrim(string A)，它返回 A 从结束修整空格产生的字符串(右侧)
string	regexp_replace(string A,	它返回从替换所有子在 B 结果配合 C.在 Java

	string B, string C)	正则表达式语法的字符串
int	size(Map<K.V>)	它返回在映射类型的元素的数量。
int	size(Array<T>)	它返回在数组类型元素的数量。
value of <type>	cast(<expr> as <type>)	它把表达式的结果 expr<类型>如 cast('1'作为 BIGINT)代表整体转换为字符串'1'。如果转换不成功，返回的是 NULL。
string	from_unixtime(int unixtime)	转换的秒数从 Unix 纪元(1970-01-0100:00:00 UTC)代表那一刻，在当前系统时区的时间戳字符的串格式："1970-01-01 00:00:00"
string	to_date(string timestamp)	返回一个字符串时间戳的日期部分：to_date("1970-01-01 00:00:00") = "1970-01-01"
int	year(string date)	返回年份部分的日期或时间戳字符串：year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970
int	month(string date)	返回日期或时间戳记字符串月份部分：month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11
int	day(string date)	返回日期或时间戳记字符串当天部分：day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1
string	get_json_object(string json_string, string path)	提取从基于指定的 JSON 路径的 JSON 字符串 JSON 对象，并返回提取的 JSON 字符串的 JSON 对象。如果输入的 JSON 字符串无效，返回 NULL。

五、视图和索引

可以创建一个视图，在执行 SELECT 语句的时候。语法如下：

```
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment],
 ...) ]

[COMMENT table_comment]

AS SELECT ...
```

示例

举个例子来看。假设 employee 表拥有如下字段：Id, Name, Salary, Designation 和 Dept。生成一个查询检索工资超过 30000 卢比的员工详细信息，我们把结果存储在一个名为视图 emp_30000.

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 40000       | Hr Admin          | HR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

下面使用上述业务情景查询检索员的工详细信息：

```
hive> CREATE VIEW emp_30000 AS
   > SELECT * FROM employee
   > WHERE salary>30000;
```

删除一个视图

使用下面的语法来删除视图：

```
DROP VIEW view_name
```

下面的查询删除一个名为 emp_30000 的视图：

```
hive> DROP VIEW emp_30000;
```

创建索引

索引也不过是一个表上的一个特定列的指针。创建索引意味着创建一个表上的一个特定列的指针。它的语法如下：

```
CREATE INDEX index_name

ON TABLE base_table_name (col_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property_name=property_value, ...)]

[IN TABLE index_table_name]

[PARTITIONED BY (col_name, ...)]

[

   [ ROW FORMAT ...] STORED AS ...

   | STORED BY ...

]

[LOCATION hdfs_path]

[TBLPROPERTIES (...)]
```

例子

让我们举个索引例子。使用之前的字段 Id, Name, Salary, Designation, 和 Dept 创建一个名为 index_salary 的索引，对 employee 表的 salary 列索引。

下面的查询创建一个索引：

```
hive> CREATE INDEX inedx_salary ON TABLE employee(salary)

   > AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';
```

这是一个指向 salary 列。如果列被修改，变更使用的索引值存储。

删除索引

下面的语法用来删除索引：

```
DROP INDEX <index_name> ON <table_name>
```

下面的查询删除名为 index_salary 索引：

```
hive> DROP INDEX index_salary ON employee;
```

六、Select where

Hive 查询语言(HiveQL)是一种查询语言,Hive 处理在 Metastore 分析结构化数据。本章介绍了如何使用 SELECT 语句的 WHERE 子句。

SELECT 语句用来从表中检索的数据。 WHERE 子句中的工作原理类似于一个条件。它使用这个条件过滤数据，并返回给出一个有限的结果。内置运算符和函数产生一个表达式，满足以下条件。

语法

下面给出的是 SELECT 查询的语法：

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]

[LIMIT number];
```

示例

让我们举个例子 SELECT ... WHERE 子句。假设 employee 表有如下 Id, Name, Salary, Designation, 和 Dept 等字段，生成一个查询检索超过 30000 薪水的员工详细信息。

```
+------+--------------+-------------+-------------------+--------+
```

```
| ID   | Name         | Salary     | Designation        | Dept   |
+------+--------------+------------+--------------------+--------+
|1201  | Gopal        | 45000      | Technical manager  | TP     |
|1202  | Manisha      | 45000      | Proofreader        | PR     |
|1203  | Masthanvali  | 40000      | Technical writer   | TP     |
|1204  | Krian        | 40000      | Hr Admin           | HR     |
|1205  | Kranthi      | 30000      | Op Admin           | Admin  |
+------+--------------+------------+--------------------+--------+
```

下面的查询检索使用上述业务情景的员工详细信息：

```
hive> SELECT * FROM employee WHERE salary>30000;
```

成功执行查询后，能看到以下回应：

```
+------+--------------+------------+--------------------+--------+
| ID   | Name         | Salary     | Designation        | Dept   |
+------+--------------+------------+--------------------+--------+
|1201  | Gopal        | 45000      | Technical manager  | TP     |
|1202  | Manisha      | 45000      | Proofreader        | PR     |
|1203  | Masthanvali  | 40000      | Technical writer   | TP     |
|1204  | Krian        | 40000      | Hr Admin           | HR     |
+------+--------------+------------+--------------------+--------+
```

七、Select order by

本章介绍了如何使用 SELECT 语句的 ORDER BY 子句。ORDER BY 子句用于检索基于一列的细节并设置排序结果按升序或降序排列。

语法

下面给出的是 ORDER BY 子句的语法：

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference [WHERE where_condition] [GROUP BY col_list] [HAVING
```

having_condition] [ORDER BY col_list]] [LIMIT number];

示例

让我们举个 SELECT ... ORDER BY 子句的例子。假设员工表，如下 Id, Name, Salary, Designation, 和 Dept 的字段，生成一个查询用于检索员工的详细信息。

ID	Name	Salary	Designation	Dept
1201	Gopal	45000	Technical manager	TP
1202	Manisha	45000	Proofreader	PR
1203	Masthanvali	40000	Technical writer	TP
1204	Krian	40000	Hr Admin	HR
1205	Kranthi	30000	Op Admin	Admin

下面是使用上述业务情景查询检索员工详细信息：

```
hive> SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
```

成功执行查询后，能看到以下回应：

ID	Name	Salary	Designation	Dept
1205	Kranthi	30000	Op Admin	Admin
1204	Krian	40000	Hr Admin	HR
1202	Manisha	45000	Proofreader	PR
1201	Gopal	45000	Technical manager	TP
1203	Masthanvali	40000	Technical writer	TP

八、Select group by

本章介绍了 SELECT 语句的 GROUP BY 子句。GROUP BY 子句用于分类所有记录结果的特定集合列。它被用来查询一组记录。

语法

GROUP BY 子句的语法如下：

SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference [WHERE where_condition] [GROUP BY col_list] [HAVING having_condition] [ORDER BY col_list]] [LIMIT number];

示例

让我们以 SELECT... GROUP BY 子句为例。假设员工表有如下 Id, Name, Salary, Designation, 和 Dept 字段。产生一个查询以检索每个部门的员工数量。

```
+------+--------------+-------------+-------------------+--------+
| ID   | Name         | Salary      | Designation       | Dept   |
+------+--------------+-------------+-------------------+--------+
|1201  | Gopal        | 45000       | Technical manager | TP     |
|1202  | Manisha      | 45000       | Proofreader       | PR     |
|1203  | Masthanvali  | 40000       | Technical writer  | TP     |
|1204  | Krian        | 45000       | Proofreader       | PR     |
|1205  | Kranthi      | 30000       | Op Admin          | Admin  |
+------+--------------+-------------+-------------------+--------+
```

下面使用上述业务情景查询检索员工的详细信息。

```
hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

成功执行查询后，能看到以下回应：

```
+------+--------------+
| Dept | Count(*)     |
+------+--------------+
|Admin |    1         |
|PR    |    2         |
|TP    |    3         |
+------+--------------+
```

九、Select JOIN

JOIN 是子句用于通过使用共同值组合来自两个表特定字段。它是用来从数据库中的两个或更多的表组合的记录。它或多或少类似于 SQL JOIN。

语法

```
join_table:

    table_reference JOIN table_factor [join_condition]
    | table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference
    join_condition
    | table_reference LEFT SEMI JOIN table_reference join_condition
    | table_reference CROSS JOIN table_reference [join_condition]
```

示例

我们在本章中将使用下面的两个表。考虑下面的表 CUSTOMERS..

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
| 1  | Ramesh   | 32  | Ahmedabad | 2000.00  |
| 2  | Khilan   | 25  | Delhi     | 1500.00  |
| 3  | kaushik  | 23  | Kota      | 2000.00  |
| 4  | Chaitali | 25  | Mumbai    | 6500.00  |
| 5  | Hardik   | 27  | Bhopal    | 8500.00  |
| 6  | Komal    | 22  | MP        | 4500.00  |
| 7  | Muffy    | 24  | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

考虑另一个表命令如下：

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 | 3000   |
| 100 | 2009-10-08 00:00:00 |           3 | 1500   |
| 101 | 2009-11-20 00:00:00 |           2 | 1560   |
| 103 | 2008-05-20 00:00:00 |           4 | 2060   |
+-----+---------------------+-------------+--------+
```

有不同类型的联接给出如下：

JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

JOIN

JOIN 子句用于合并和检索来自多个表中的记录。 JOIN 和 SQLOUTER JOIN 类似。连接条件是使用主键和表的外键。

下面的查询执行 JOIN 的 CUSTOMER 和 ORDER 表，并检索记录：

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
   > FROM CUSTOMERS c JOIN ORDERS o
   > ON (c.ID = o.CUSTOMER_ID);
```

成功执行查询后，能看到以下回应：

```
+----+----------+-----+--------+
```

```
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

LEFT OUTER JOIN

HiveQL LEFT OUTER JOIN 返回所有行左表，即使是在正确的表中没有匹配。这意味着，如果 ON 子句匹配的右表 0(零)记录,JOIN 还是返回结果行,但在右表中的每一列为 NULL。LEFT JOIN 返回左表中的所有的值，加上右表，或 JOIN 子句没有匹配的情况下返回 NULL。

下面的查询演示了 CUSTOMER 和 ORDER 表之间的 LEFT OUTER JOIN 用法：

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
    > FROM CUSTOMERS c
    > LEFT OUTER JOIN ORDERS o
    > ON (c.ID = o.CUSTOMER_ID);
```

成功执行查询后，能看到以下回应：

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

RIGHT OUTER JOIN

HiveQL RIGHT OUTER JOIN 返回右边表的所有行，即使有在左表中没有匹配。如果 ON 子句的左表匹配 0（零）的记录，JOIN 结果返回一行，但在左表中的每一列为 NULL。

RIGHT JOIN 返回右表中的所有值，加上左表，或者没有匹配的情况下返回 NULL。

下面的查询演示了在 CUSTOMER 和 ORDER 表之间使用 RIGHT OUTER JOIN。

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
```

```
    > FROM CUSTOMERS c
    > RIGHT OUTER JOIN ORDERS o
    > ON (c.ID = o.CUSTOMER_ID);
```

成功执行查询后，能看到以下回应：

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

FULL OUTER JOIN

HiveQL FULL OUTER JOIN 结合了左边，并且满足 JOIN 条件合适外部表的记录。连接表包含两个表的所有记录，或两侧缺少匹配结果那么使用 NULL 值填补

下面的查询演示了 CUSTOMER 和 ORDER 表之间使用的 FULL OUTER JOIN：

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
    > FROM CUSTOMERS c
    > FULL OUTER JOIN ORDERS o
    > ON (c.ID = o.CUSTOMER_ID);
```

成功执行查询后，能看到以下回应：

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
| 1    | Ramesh   | NULL   | NULL                |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5    | Hardik   | NULL   | NULL                |
| 6    | Komal    | NULL   | NULL                |
| 7    | Muffy    | NULL   | NULL                |
| 3    | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3    | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 2    | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 4    | Chaitali | 2060   | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```