Hive常用操作

第01节 DDL基础

1. 建表语法概述

Hive建表完整语法如下：

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type [column_constraint_specification] [COMMENT col_comment],
... [constraint_specification])]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...) [SORTED BY (col_name [ASC|DESC], ...)]
INTO num_buckets BUCKETS]
  [SKEWED BY (col_name, col_name, ...)
   ON ((col_value, col_value, ...), (col_value, col_value, ...), ...)
   [STORED AS DIRECTORIES]
   [
       [ROW FORMAT row_format]
       [STORED AS file_format]
       | STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)]
   ]
   [LOCATION hdfs_path]
   [TBLPROPERTIES (property_name=property_value, ...)]
   [AS select_statement];
```

详细语法参考文档：https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Overview

- CREATE TABLE，创建一个指定名称的表，如果存在同名表，则抛出异常，可以使用IF NOT EXISTS忽略异常
- EXTERNAL：表示该表为外部表，须要同时指定实际数据的HDFS路径LOCATION
- TEMPORARY：表示该表为临时表，临时表只对当前会话有效，会话退出时会被自动删除，临时表不支持分区与索引
- PARTITIONED BY：创建表的时候可以同时为表指定多个分区
- CLUSTERED BY：分桶，让数据能够均匀地分布在表的各个数据文件中

接下来针对上述语法中的重点部分进行详细说明。

2. 数据类型

Hive中的数据类型指的是Hive表中的列字段类型，Hive数据类型整体分为两个类别，基本数据类型和复杂数据类型。

基本数据类型包括：数值类型（整型，浮点型，定点数）、时间类型、字符串类型、杂项数据类型（布尔型，二进制型）；

复杂数据类型包括：array数组、map映射、struct结构、union联合体；

基本数据类型

大类	类型
Integers（整型）	TINYINT—1 字节的有符号整数 SMALLINT—2 字节的有符号整数 INT—4 字节的有符号整数 BIGINT—8 字节的有符号整数 使用整数，默认是INT，其他整型需要加后缀，TINYINT、SMALLINT、BIGINT后缀为Y、S、L，如100Y，100S，100L
Floating point numbers（浮点型）	FLOAT— 单精度浮点型 DOUBLE—双精度浮点型
Fixed point numbers（定点数）	用户自定义精度定点数，DECIMAL(precision[,scale])，precision表示固定精度，最大为38，scale表示小数位数，如DECIMAL(5,2)，表示-999.99到999.99，DECIMAL(5)表示，-99999到99999，DECIMAL等同于DECIMAL(10,0)
String types（字符串）	STRING—存储可变长文本，对长度没有限制 VARCHAR—具有最大长度限制的字符序列 CHAR—固定长度的字符序列
Date and time types（日期时间类型）	TIMESTAMP — 时间戳 TIMESTAMP WITH LOCAL TIME ZONE — 时间戳，纳秒精度 DATE—日期类型，以yyyy-MM-dd表示
Boolean（布尔型）	BOOLEAN—TRUE/FALSE
Binary types（二进制类型）	BINARY—字节序列

上述数据类型都是对Java中的接口的实现，所以类型的具体行为细节和Java中对应的类型一致，如STRING对象Java中的String，FLOAT对象Java中的float。

需要注意：

- HQL对大小写不敏感
- Hive中所有类型都是保留字

TIMESTAMP 和 TIMESTAMP WITH LOCAL TIME ZONE 的区别如下：

- **TIMESTAMP WITH LOCAL TIME ZONE**：用户提交时间给数据库时，会被转换成数据库所在的时区来保存。查询时则按照查询客户端的不同，转换为查询客户端所在时区的时间。
- **TIMESTAMP** ：提交什么时间就保存什么时间，查询时也不做任何转换。

复杂数据类型

在SQL的表设计中，字段通常不能再被分解，也就是一个字段不能再被分割成多个字段，如下表中的订单是不满足第一范式的。

订单编号	商品项
1000	["苹果","西瓜","橘子"]

与SQL不同，HQL没有上述限制，Hive表中的字段不仅可以是基本数据类型，也可以是复杂数据类型，具体类型如下：

类型	描述	示例
STRUCT	类似于对象，是字段的集合，字段的类型可以不同，可以使用 名称.字段名 方式进行访问	数据格式：{'张三',22} 定义示例： struct<name:string,age:int>
MAP	键值对的集合，可以使用 名称[key] 的方式访问对应的值	数据格式：{'A':'Apple','O':'Orange'} 定义示例：map<string,string>
ARRAY	数组是一组具有相同类型和名称的变量的集合，可以使用名称[index] 访问对应的值	数据格式：[10,20,30] 定义示例：array
UNION	类似于Java中的泛型，任一时刻只有其中一种类型生效	定义示例 uniontype<data_type,data_type...> 可以通过create_union内置函数创建uniontype：create_union(tag, val1, val2)

UNIONTYPE数据类型是在Hive 0.7.0中引入的，但在Hive中对该类型的完全支持仍然不完整，在JOIN、WHERE和GROUP BY子句中引用UNIONTYPE字段的查询将会失败。

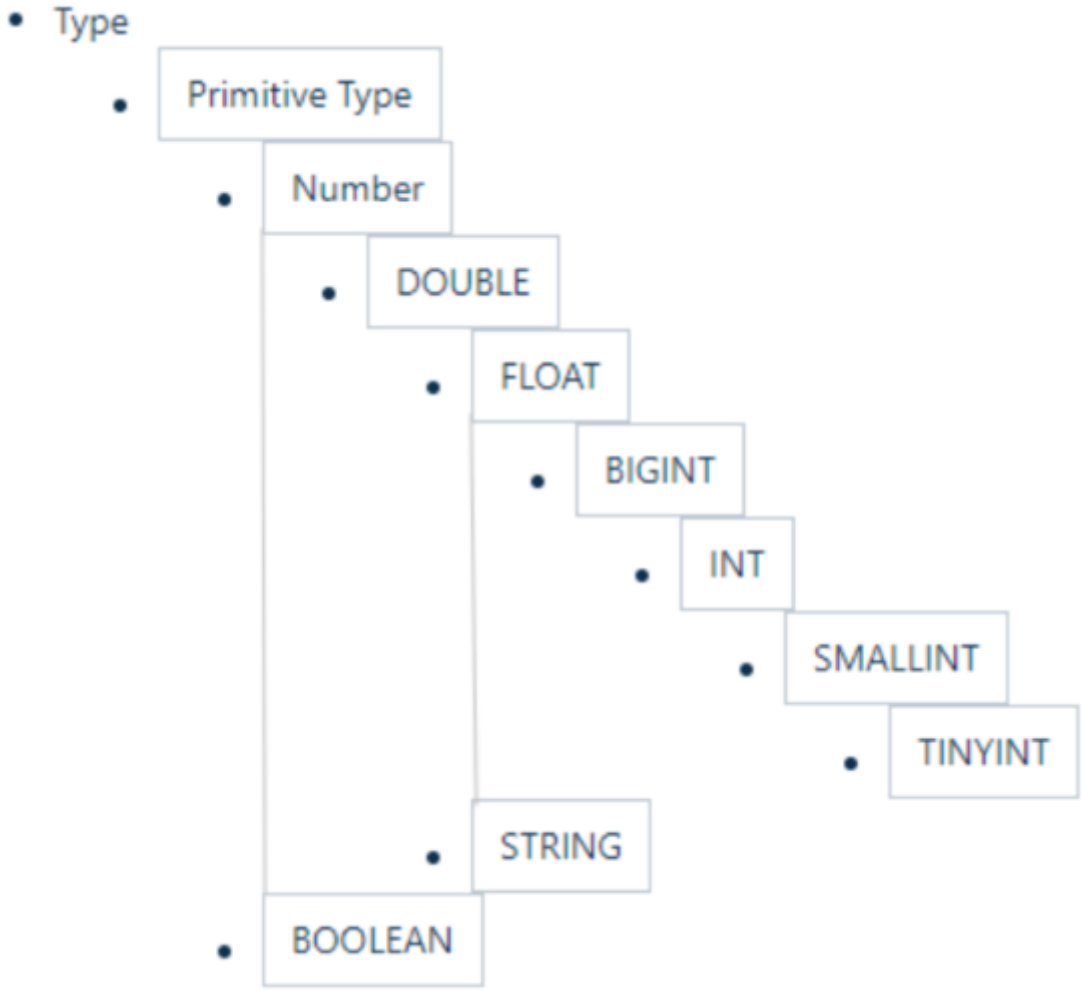
建表示例

```
CREATE TABLE students(
    name     STRING,    -- 姓名
    age        BIGINT,       -- 年龄
    subject   ARRAY<STRING>,    -- 学科
    score     MAP<STRING,FLOAT>,  -- 各个学科考试成绩
    address   STRUCT<houseNumber:int, street:STRING, citySTRING, province:
    STRING>  -- 家庭居住地址
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY "\t";
```

详细语法参考文档：：https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types

3. 数据类型转换

与SQL类似，HQL支持隐式和显式类型转换，Hive 中基本数据类型遵循以下的层次结构，按照这个层次结构，子类型到祖先类型允许隐式转换。例如 INT 类型的数据允许隐式转换为 BIGINT 类型。额外注意的是：按照类型层次结构允许将 STRING 类型隐式转换为 DOUBLE类型。



显示类型转换可以使用CAST函数，例如：CAST（'100'as INT）会将100字符串转换为100整数值。 如果强制转换失败，例如CAST（'INT'as INT），该函数返回NULL。

```
hive> select cast(12.1 as int);
OK
12
Time taken: 0.487 seconds, Fetched: 1 row(s)
hive>
```

4. 内容格式

当数据存储在文本文件中，必须按照一定格式区别行和列，如使用逗号作为分隔符的 CSV 文件(Comma-Separated Values) 或者使用制表符作为分隔值的 TSV 文件 (Tab-Separated Values)。但此时也存在一个缺点，就是正常的文件内容中也可能出现逗号或者制表符。

所以 Hive 默认使用了几个平时很少出现的字符，这些字符一般不会作为内容出现在文件中。Hive 默认的行和列分隔符如下表所示。

分隔符	描述
\n	对于文本文件来说，每行是一条记录，所以可以使用换行符来分割记录
^A	分割字段 (列)，在 CREATE TABLE 语句中也可以使用八进制编码 \001 来表示
^B	用于分割 ARRAY 或者 STRUCT 中的元素，或者用于 MAP 中键值对之间的分割，在 CREATE TABLE 语句中也可以使用八进制编码 \002 表示
^C	用于 MAP 中键和值之间的分割，在 CREATE TABLE 语句中也可以使用八进制编码\003 表示

使用示例如下：

```
CREATE TABLE page_view(
    viewTime INT,
    userid BIGINT
)
 ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\001'
   COLLECTION ITEMS TERMINATED BY '\002'
   MAP KEYS TERMINATED BY '\003'
 STORED AS SEQUENCEFILE;
```

ROW FORMAT

ROW FORMAT用于指定序列化与反序列化器（Serializer/Deserializer，简称SerDe），Hive使用SerDe读写表的每一行数据，如果ROW FORMAT未指定，或者指定为“ROW FORMAT **DELIMITED**”，则表示Hive使用内置的SerDe（默认为LazySimpleSerDe）来处理每行数据。

如果数据文件格式比较特殊可以使用ROW FORMAT **SERDE** serde_name指定其他的Serde类来处理数据,甚至支持用户自定义SerDe类。

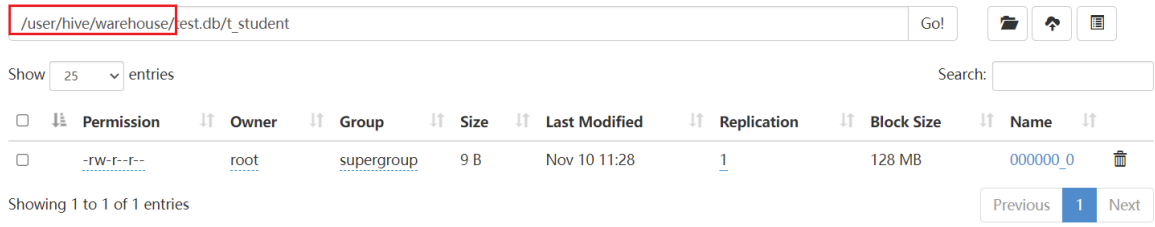
上述示例中，因为指定了“ROW FORMAT DELIMITED”，Hive将使用内置的SerDe处理数据，内置的SerDe还可详细指定字段之间、集合元素之间、map映射 kv之间、换行的分隔符号，具体如下：

ROW FORMAT **DELIMITED**

- FIELDS TERMINATED BY：用于指定字段之间分隔符
- COLLECTION ITEMS TERMINATED BY：用于指定集合元素之间分隔符
- MAP KEYS TERMINATED BY：用于指定MapKV之间的分隔符
- LINES TERMINATED BY：用于指定行数据之间的分隔符

5. 数据存储路径

Hive表默认存储路径是由${HIVE_HOME}/conf/hive-site.xml配置文件的hive.metastore.warehouse.dir属性指定，如果未指定，其默认值为：/user/hive/warehouse



在Hive建表的时候，可以通过location语法来更改数据在HDFS上的存储路径。

语法：[LOCATION hdfs_path]

对于已经生成好的数据文件，使用location指定路径将会很方便。

第02节 数据库及表常见操作

1. 数据库操作

Create Database

Hive中DATABASE和SCHEMA代表的都是数据库，可以互换。

创建数据库语法如下：

```
CREATE [REMOTE] (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
  [COMMENT database_comment]  -- 数据库注释
  [LOCATION hdfs_path]        -- 指定数据库在HDFS的存储位置，默认
在/user/hive/warehouse
  [WITH DBPROPERTIES (property_name=property_value, ...)]; -- 指定数据库属性配置
```

示例：

```
create database if not exists mydb;
```

Drop Database

删除数据语法如下：

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

删除数据库时默认为RESTRICT，此时数据库必须是空的，如果数据库中有表则需要使用CASCADE。

示例：

```
drop database mydb;
```

Alter Database

Hive中的ALTER DATABASE语句用于更改与Hive中的数据库关联的元数据，语法如下：

```
-- 修改数据库属性配置
ALTER (DATABASE|SCHEMA) database_name SET DBPROPERTIES
(property_name=property_value, ...);
-- 修改数据库所有者
ALTER (DATABASE|SCHEMA) database_name SET OWNER [USER|ROLE] user_or_role;
-- 修改数据库位置
ALTER (DATABASE|SCHEMA) database_name SET LOCATION hdfs_path;
```

Use Database

USE DATABASE语句用于选择特定的数据库,切换当前会话使用哪一个数据库进行操作,语法如下:

```
USE database_name;
```

Describe Database

Describe Database语句用于显示数据库相关元信息，如数据库名称、位置、所有者等，语法如下:

```
DESCRIBE DATABASE [EXTENDED] db_name;
```

EXTENDED用于显示更多额外信息。

2. 表操作

Drop Table

DROP TABLE删除该表的元数据和数据，如果已配置垃圾桶（且未指定PURGE），则该表对应的数据实际上将移动到.Trash/Current目录，而元数据完全丢失。

如果指定了PURGE，则表数据不会进入.Trash/Current目录，跳过垃圾桶直接被删除。

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

Truncate Table

从表中删除所有行。可以简单理解为清空表的所有数据但是保留表的元数据结构。如果HDFS启用了垃圾桶，数据将被丢进垃圾桶，否则将被删除。

```
TRUNCATE [TABLE] table_name;
```

Describe Table

Describe Table用于显示表的元数据信息，语法如下:

```
DESCRIBE [EXTENDED|FORMATTED] table_name;
```

EXTENDED与FORMATTED会以不同的信息显示表格信息。

Alter Table

Alter Table用于修改表，具体使用方式如下:

- 修改表名

```
ALTER TABLE table_name RENAME TO new_table_name;
```

- 更改SerDe属性

```
ALTER TABLE table_name [PARTITION partition_spec] SET SERDEPROPERTIES
serde_properties;
serde_properties:
  : (property_name = property_value, property_name = property_value, ... )
```

示例：

```
ALTER TABLE table_name SET SERDEPROPERTIES ('field.delim' = ',');
```

- 修改表的存储位置

```
ALTER TABLE table_name SET LOCATION "new location";
```

- 修改列

```
ALTER TABLE table_name [PARTITION partition_spec] CHANGE [COLUMN]
col_old_name col_new_name column_type
  [COMMENT col_comment] [FIRST|AFTER column_name] [CASCADE|RESTRICT];
```

官方示例：

```
-- 创建示例表
CREATE TABLE test_change (a int, b int, c int);

-- First change column a's name to a1.
ALTER TABLE test_change CHANGE a a1 INT;

-- Next change column a1's name to a2, its data type to string, and put it
after column b.
ALTER TABLE test_change CHANGE a1 a2 STRING AFTER b;
-- The new table's structure is:  b int, a2 string, c int.

-- Then change column c's name to c1, and put it as the first column.
ALTER TABLE test_change CHANGE c c1 INT FIRST;
-- The new table's structure is:  c1 int, b int, a2 string.

-- Add a comment to column a1
ALTER TABLE test_change CHANGE a1 a1 INT COMMENT 'this is column a1';
```

3. 显示信息

- 显示所有数据库

```
show databases;
```

- 显示当前数据库所有表

```
show tables;
```

- 显示表分区信息

```
show partitions table_name;
```

- 显示表创建语句

```
show create table table_name;
```

- 显示表中的所有列，包括分区列

```
show columns in table_name;
```

第03节 加载数据

1. LOAD DATA

在表创建好以后，不论是分区表还是非分区表都可以使用load将数据加载到表中，加载时，Hive不会进行任何转换，仅将数据文件移动到Hive表对应的目录中。

```
LOAD DATA [LOCAL] INPATH 'filepath'
[OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

示例：

```
-- 往某非分区表加载数据
load data local inpath  '/root/covid/2020-02.csv' into table covid19_all;
-- 加载数据到分区
load data inpath '/data/covid/2020-02-20.csv' into table covid19
partition(dt='2020-02-20');
```

- 如果使用 OVERWRITE 关键字，则将删除目标表（或分区）的内容，使用新的数据填充；不使用此关键字，则数据以追加的方式加入
- LOCAL 关键字代表从本地文件系统加载文件，省略则代表从 HDFS 上加载文件：
- 加载的目标可以是表或分区。如果是分区表，则必须指定加载数据的分区；

2. INSERT+ SELECT

由于数据仓库的数据主要来源于各业务系统，因此很少使用insert+values的形式插入数据，既：

```
INSERT INTO table_name ( field1, field2,...fieldN )
VALUES
( value1, value2,...valueN );
```

示例：

```
CREATE TABLE students (name VARCHAR(64), age INT, gpa DECIMAL(3, 2));
INSERT INTO TABLE students
VALUES ('fred flintstone', 35, 1.28), ('barney rubble', 32, 2.32);
```

上述方式在关系型数据库中很常见，数据仓库中经常使用insert + select的方式加载数据，也就是将select查询出的数据，插入到表中。

具体语法：

```
-- 以覆盖的形式插入
INSERT OVERWRITE TABLE tablename1  select_statement1 FROM from_statement;

-- 以追加的形式插入
INSERT INTO TABLE tablename1  select_statement1 FROM from_statement;
```

- 要保证查询结果列的数目和需要插入数据表格的列数目一致
- 如果查询出来的数据类型和插入表格对应的列数据类型不一致，将会进行转换，但是不能保证转换一定成功，转换失败的数据将会为NULL
- 从 Hive 1.2.0 开始，可以采用 INSERT INTO tablename(z，x，c1) 指明插入列

为了提高效率，Hive支持将 SELECT 语句的查询结果插入多个表（或分区），称为多表插入，可以理解为一次读取，多次插入，具体语法：

```
FROM from_statement
INSERT OVERWRITE TABLE tablename1
select_statement1
[INSERT OVERWRITE TABLE tablename2
select_statement2]
[INSERT INTO TABLE tablename2 select_statement2] ...;
```

第04节 查询数据

Hive查询语法如下：

```
[WITH CommonTableExpression (, CommonTableExpression)*]
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
  FROM table_reference  -- 可以是表，视图，join查询结果或子查询结果
  [WHERE where_condition]
  [GROUP BY col_list]
  [ORDER BY col_list]
  [CLUSTER BY col_list
    | [DISTRIBUTE BY col_list] [SORT BY col_list]
  ]
 [LIMIT [offset,] rows]
```

基础查询部分，基于之前的非分区表covid19_all来演示。

1. 基础查询

- 按字段查询

```
-- 查询所有字段
select * from covid19_all;
-- 指定字段查询
select county, cases, deaths from covid19_all;
```

- 去重，ALL和DISTINCT选项指定是否应返回重复的行。如果没有给出这些选项，则默认值为ALL（返回所有匹配的行）,DISTINCT指定从结果集中删除重复的行。

```
select distinct state from covid19_all;
```

- 条件查询，WHERE条件是一个布尔表达式。在WHERE表达式中，可以使用Hive支持的任何函数和运算符，但聚合函数除外。

```
select * from covid19_all where state ="Oregon" and cases  > 0;

-- where条件中使用函数 找出名称大于5个字符的州
select * from covid19_all where length(state) > 5;

-- where条件中不能使用聚合函数
select state,sum(deaths) from covid19_all where sum(cases) > 10 group by
state;
```

- 分区查询，第02节中创建的分区表covid19_p，以count_date为分区字段，对分区表进行查询时，Hive会检查WHERE子句或JOIN中的ON子句中是否存在对分区字段的过滤，如果存在，则仅访问查询符合条件的分区,避免全表扫描

```
select * from covid19_p where count_date > '2020-02-10' and count_date <
'2020-02-15';
```

- 分组查询，Hive 支持使用 GROUP BY 进行分组聚合操作。

```
set hive.map.aggr=true;
-- 查询各个部门薪酬综合
SELECT state,SUM(cases) FROM covid19_all GROUP BY state;
```

  hive.map.aggr 控制程序如何进行聚合。默认值为 false。如果设置为 true，Hive 会在 map 阶段就执行一次聚合。这可以提高聚合效率，但需要消耗更多内存。

  注意，出现在GROUP BY中select_expr的字段，除被聚合函数应用的字段外，其他的必须是被 group by 分组的字段，否则会出现错误，如：

```
-- SemanticException [Error 10025]: Line 1:7 Expression not in GROUP BY key
'county'
SELECT county,state,SUM(cases) FROM covid19_all GROUP BY state;
```

- HAVING，WHERE关键字无法与聚合函数一起使用，但可以使用 HAVING 对分组数据进行过滤，之所以HAVING能够生效是因为HAVING执行时，where，group by已经执行结束。

```
select state,sum(cases) from covid19_all group by state having sum(cases) >
10;
-- 可以再select_expr中使用别名的方式，然后再having子句中直接使用别名引用之前计算的结果，
提高运算效率
select state,sum(cases) as cases_all from covid19_all group by state having
cases_all > 10;
```

having与where的区别：

- having是在分组后对数据进行过滤
- where是在分组前对数据进行过滤
- having后面可以使用聚合函数
- where后面不可以使用聚合

- LIMIT，LIMIT子句可用于约束SELECT语句返回的行数，第一个参数指定要返回的第一行的偏移量（从 Hive 2.0.0开始），第二个参数指定要返回的最大行数。当给出单个参数时，它代表最大行数，并且偏移量默认为0。

```
-- 返回结果集的前5条
select * from covid19_all where count_date = "2020-02-22"  limit 5;

-- 返回结果集从第2行开始 共2行
select * from covid19_all where count_date = "2020-02-22"  limit 1,2;
```

2. 复杂查询

ORDER BY [ASC|DESC]

可以使用 ORDER BY 或者 Sort BY 对查询结果进行排序，排序字段可以是整型也可以是字符串：如果是整型，则按照大小排序；如果是字符串，则按照字典序排序。ORDER BY 和 SORT BY 的区别如下：

- 使用 ORDER BY 时会有一个 Reducer 对全部查询结果进行排序，可以保证数据的全局有序性；
- 使用 SORT BY 时只会在每个 Reducer 中进行排序，这可以保证每个 Reducer 的输出数据是有序的，但不能保证全局有序

```
-- 根据字段进行排序
select * from covid19_all where count_date = "2020-02-22"  order by cases;
select * from covid19_all where count_date = "2020-02-22"  order by cases desc;
```

由于 ORDER BY 的时间可能很长，如果你设置了严格模式 (hive.mapred.mode = strict)，则其后面必须再跟一个 limit 子句，hive.mapred.mode 默认值是 nonstrict ，也就是非严格模式，议将LIMIT与ORDER BY一起使用，避免数据集行数过大。

```
select * from covid19_all where count_date = "2020-02-22"  order by cases desc
limit 3;
```

Union联合查询

UNION用于将来自多个SELECT语句的结果合并为一个结果集。语法如下：

```
select_statement UNION [ALL | DISTINCT]
select_statement UNION [ALL | DISTINCT]
select_statement ...
```

使用DISTINCT关键字与只使用UNION默认值效果一样，都会删除重复行。

使用ALL关键字，不会删除重复行，结果集包括所有SELECT语句的匹配行（包括重复行）。

注意：每个select_statement返回的列的数量和名称必须相同

如果要将ORDER BY，SORT BY，CLUSTER BY，DISTRIBUTE BY或LIMIT应用于单个SELECT，需要将子句放在括住SELECT的括号内：

```
SELECT key FROM (SELECT key FROM src ORDER BY key LIMIT 10)subq1
UNION
SELECT key FROM (SELECT key FROM src1 ORDER BY key LIMIT 10)subq2
```

如果要将ORDER BY，SORT BY，CLUSTER BY，DISTRIBUTE BY或LIMIT子句应用于整个UNION结果，需要将这些关键字放到最后：

```
SELECT key FROM src
UNION
SELECT key FROM src1
ORDER BY key LIMIT 10
```

3. 子查询

子查询语法：

```
SELECT ... FROM (subquery) name ...
SELECT ... FROM (subquery) AS name ...  -- Hive 0.13.0和更高版本中的子查询名称之前可以包含可选关键字“ AS”
```

示例：

```
SELECT col
FROM (
  SELECT a+b AS col
  FROM t1
) t2
```

包含 UNION ALL 的子查询：

```
SELECT t3.col
FROM (
  SELECT a+b AS col
  FROM t1
  UNION ALL
  SELECT c+d AS col
  FROM t2
) t3
```

在where子句中使用子查询，查询结果可以被视为IN和NOT IN语句的常量，注意此时子查询的结果只能是一列。

```
SELECT *
FROM A
WHERE A.a IN (SELECT foo FROM B);
```

4. 连接查询

数据准备

为了演示连接查询操作，这里需要预先创建两张表，并加载测试数据。

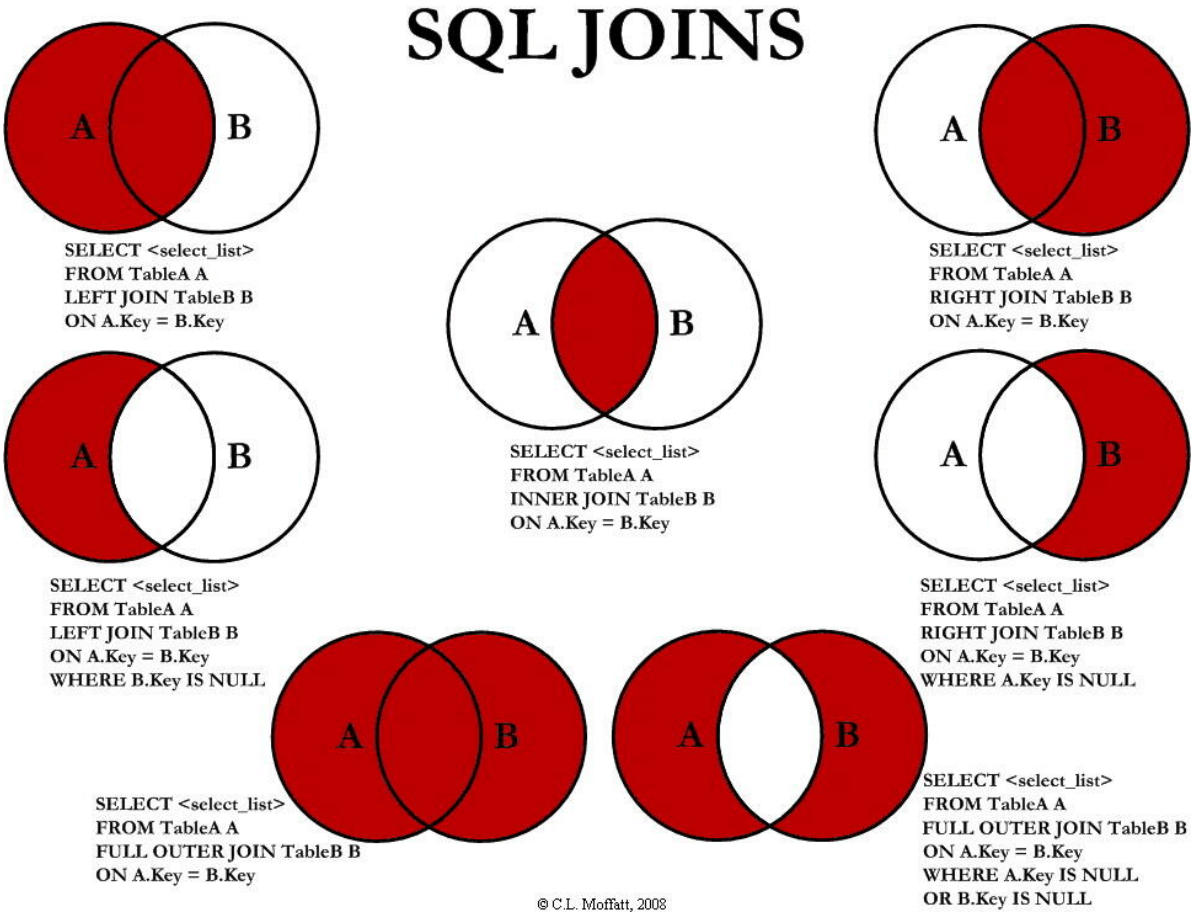
员工表：

```
-- 建表语句
CREATE TABLE emp(
    empno INT,       -- 员工表编号
    ename STRING,    -- 员工姓名
    job STRING,      -- 职位类型
    mgr INT,         -- 上级编号
    hiredate TIMESTAMP, -- 雇佣日期
    sal DECIMAL(7,2),   -- 工资
    comm DECIMAL(7,2),
    deptno INT)           -- 部门编号
    ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
-- 加载数据
LOAD DATA LOCAL INPATH "/root/data/emp.txt" OVERWRITE INTO TABLE emp;
```

部门表：

```
-- 建表语句
CREATE TABLE dept(
    deptno INT,    -- 部门编号
    dname STRING,  -- 部门名称
    loc STRING     -- 部门所在的城市
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY "\t";
-- 加载数据
LOAD DATA LOCAL INPATH "/root/data/dept.txt" OVERWRITE INTO TABLE dept;
```

Hive 支持内连接，外连接，左外连接，右外连接，笛卡尔连接，这和传统数据库中的概念是一致的，可以参见下图。



需要特别强调：JOIN 语句的关联条件必须用 ON 指定，不能用 WHERE 指定，否则就会先做笛卡尔积，再过滤，这会导致你得不到预期的结果 (下面的演示会有说明)。

INNER JOIN

```
-- 查询员工编号为 7369 的员工的详细信息
SELECT e.*,d.*
FROM emp e
JOIN dept d
ON e.deptno = d.deptno
WHERE empno = 7369;
-- 如果是三表或者更多表连接，语法如下
SELECT a.val, b.val, c.val
FROM a
JOIN b ON (a.key = b.key1)
JOIN c ON (c.key = b.key1)
```

LEFT OUTER JOIN

LEFT OUTER JOIN 和 LEFT JOIN 是等价的。

```
-- 左连接
SELECT e.*,d.*
FROM emp e LEFT OUTER
JOIN dept d
ON e.deptno = d.deptno;
```

RIGHT OUTER JOIN

```
-- 右连接
SELECT e.*,d.*
FROM emp e RIGHT OUTER
JOIN dept d
ON e.deptno = d.deptno;
```

执行右连接后，由于 40 号部门下没有任何员工，所以此时员工信息为 NULL。这个查询可以很好的复述上面提到的——JOIN 语句的关联条件必须用 ON 指定，不能用 WHERE 指定。你可以把 ON 改成 WHERE，你会发现无论如何都查不出 40 号部门这条数据，因为笛卡尔运算不会有 (NULL, 40) 这种情况。

```
OK
7782    CLARK   MANAGER 7839    1981-06-09 00:00:00     2450.00 NULL    10      10      ACCOUNTING      NEW YORK
7839    KING    PRESIDENT       NULL    1981-11-17 00:00:00     5000.00 NULL    10      10      ACCOUNTING      NEW YORK
7934    MILLER  CLERK   7782    1982-01-23 00:00:00     1300.00 NULL    10      10      ACCOUNTING      NEW YORK
7369    SMITH   CLERK   7902    1980-12-17 00:00:00     800.00  NULL    20      20      RESEARCH        DALLAS
7566    JONES   MANAGER 7839    1981-04-02 00:00:00     2975.00 NULL    20      20      RESEARCH        DALLAS
7788    SCOTT   ANALYST 7566    1987-04-19 00:00:00     1500.00 NULL    20      20      RESEARCH        DALLAS
7876    ADAMS   CLERK   7788    1987-05-23 00:00:00     1100.00 NULL    20      20      RESEARCH        DALLAS
7902    FORD    ANALYST 7566    1981-12-03 00:00:00     3000.00 NULL    20      20      RESEARCH        DALLAS
7499    ALLEN   SALESMAN        7698    1981-02-20 00:00:00     1600.00 300.00  30      30      SALES   CHICAGO
7521    WARD    SALESMAN        7698    1981-02-22 00:00:00     1250.00 500.00  30      30      SALES   CHICAGO
7654    MARTIN  SALESMAN        7698    1981-09-28 00:00:00     1250.00 1400.00 30      30      SALES   CHICAGO
7698    BLAKE   MANAGER 7839    1981-05-01 00:00:00     2850.00 NULL    30      30      SALES   CHICAGO
7844    TURNER  SALESMAN        7698    1981-09-08 00:00:00     1500.00 0.00    30      30      SALES   CHICAGO
7900    JAMES   CLERK   7698    1981-12-03 00:00:00     950.00  NULL    30      30      SALES   CHICAGO
NULL    NULL    NULL    NULL    NULL    NULL    NULL    40      OPERATIONS      BOSTON
Time taken: 17.138 seconds, Fetched: 15 row(s)
```

使用where的情况

```
SELECT e.*,d.*
FROM emp e RIGHT OUTER
JOIN dept d
where e.deptno = d.deptno;
```

```
OK
7369    SMITH   CLERK   7902    1980-12-17 00:00:00     800.00  NULL    20      20      RESEARCH        DALLAS
7499    ALLEN   SALESMAN        7698    1981-02-20 00:00:00     1600.00 300.00  30      30      SALES   CHICAGO
7521    WARD    SALESMAN        7698    1981-02-22 00:00:00     1250.00 500.00  30      30      SALES   CHICAGO
7566    JONES   MANAGER 7839    1981-04-02 00:00:00     2975.00 NULL    20      20      RESEARCH        DALLAS
7654    MARTIN  SALESMAN        7698    1981-09-28 00:00:00     1250.00 1400.00 30      30      SALES   CHICAGO
7698    BLAKE   MANAGER 7839    1981-05-01 00:00:00     2850.00 NULL    30      30      SALES   CHICAGO
7782    CLARK   MANAGER 7839    1981-06-09 00:00:00     2450.00 NULL    10      10      ACCOUNTING      NEW YORK
7788    SCOTT   ANALYST 7566    1987-04-19 00:00:00     1500.00 NULL    20      20      RESEARCH        DALLAS
7839    KING    PRESIDENT       NULL    1981-11-17 00:00:00     5000.00 NULL    10      10      ACCOUNTING      NEW YORK
7844    TURNER  SALESMAN        7698    1981-09-08 00:00:00     1500.00 0.00    30      30      SALES   CHICAGO
7876    ADAMS   CLERK   7788    1987-05-23 00:00:00     1100.00 NULL    20      20      RESEARCH        DALLAS
7900    JAMES   CLERK   7698    1981-12-03 00:00:00     950.00  NULL    30      30      SALES   CHICAGO
7902    FORD    ANALYST 7566    1981-12-03 00:00:00     3000.00 NULL    20      20      RESEARCH        DALLAS
7934    MILLER  CLERK   7782    1982-01-23 00:00:00     1300.00 NULL    10      10      ACCOUNTING      NEW YORK
Time taken: 17.649 seconds, Fetched: 14 row(s)
```

FULL OUTER JOIN

```
SELECT e.*,d.*
FROM emp e FULL OUTER JOIN dept d
ON e.deptno = d.deptno;
```

LEFT SEMI JOIN

LEFT SEMI JOIN （左半连接）是 IN/EXISTS 子查询的一种更高效的实现。

- JOIN 子句中右边的表只能在 ON 子句中设置过滤条件;
- 查询结果只包含左边表的数据，所以只能 SELECT 左表中的列。

```
-- 查询在纽约办公的所有员工信息
SELECT emp.*
FROM emp LEFT SEMI JOIN dept
ON emp.deptno = dept.deptno AND dept.loc="NEW YORK";
-- 上面的语句就等价于
SELECT emp.* FROM emp
WHERE emp.deptno IN (SELECT deptno FROM dept WHERE loc="NEW YORK");
```

JOIN

笛卡尔积连接，这个连接日常的开发中可能很少遇到，且性能消耗比较大，基于这个原因，如果在严格模式下 (hive.mapred.mode = strict)，Hive 会阻止用户执行此操作。

```
SELECT * FROM emp JOIN dept; -- 笛卡儿积
```

第05节 Hive内置函数

Hive SQL也内建了不少函数，满足于用户在不同场合下的数据分析需求，提高开发SQL数据分析的效率。

Hive的函数很多，除了自己内置所支持的函数之外，还支持用户自己定义开发函数。
针对内置的函数，可以根据函数的应用类型进行归纳分类，比如：数值类型函数、日期类型函数、字符串类型函数、集合函数、条件函数等；
针对用户自定义函数，可以根据函数的输入输出行数进行分类，比如：UDF、UDAF、UDTF。

1. 数学函数

返回值类型	函数名称	描述
DOUBLE	round(DOUBLE a)或round(DOUBLE a, INT d)	取整
BIGINT	floor(DOUBLE a)	向下取整
BIGINT	ceil(DOUBLE a), ceiling(DOUBLE a)	向上取整
DOUBLE	rand(), rand(INT seed)	随机数
DOUBLE	abs(DOUBLE a)	绝对值

示例：

```
-- 遵循四舍五入
select round(3.141); -- 3
-- 指定精度
select round(3.14159,2); -- 3.14
-- 取整函数
select floor(3.1415); -- 3
select floor(-3.1415); -- -4
select ceil(3.1415); -- 4
select ceil(-3.1415); -- -3

-- 返回一个0到1范围内的随机数
select rand(); -- 0.31765744631261617 每次值不同
```

```
-- 指定种子得到一个稳定的随机数序列
select rand(2);

-- 绝对值函数
select abs(-3.9); -- 3.9
```

2. 日期函数

主要针对时间、日期数据类型进行操作：

返回值类型	函数名称	描述
string	from_unixtime(bigint unixtime[, string format])	UNIX时间戳转日期函数
bigint	unix_timestamp()	获取当前UNIX时间戳函数
bigint	unix_timestamp(string date) 或 unix_timestamp(string date, string pattern)	日期转UNIX时间戳函数，可以指定具体格式

返回值类型	函数名称	描述
pre 2.1.0: string *2.1.0 on:* date	to_date(string timestamp)	抽取日期函数
int	year(string date)	日期转年函数
int	quarter(date/timestamp/string)	日期转季度函数
int	month(string date)	日期转月函数
int	day(string date) 或 dayofmonth(date)	日期转天函数
int	hour(string date)	日期转小时函数
int	minute(string date)	日期转分钟函数
int	second(string date)	日期转秒函数
int	weekofyear(string date)	日期转周函数，返回指定日期所示年份第几周
int	datediff(string enddate, string startdate)	日期比较函数
pre 2.1.0: string *2.1.0 on:* date	date_add(date/timestamp/string startdate, tinyint/smallint/int days)	日期增加函数
pre 2.1.0: string *2.1.0 on:* date	date_sub(date/timestamp/string startdate, tinyint/smallint/int days)	日期减少函数
date	current_date	获取当前日期
timestamp	current_timestamp	获取当前时间戳
string	date_format(date/timestamp/string ts, string fmt)	日期格式化

示例

```
-- 获取当前日期
select current_date(); -- 2021-11-18

-- 获取当前时间戳
-- 同一查询中对current_timestamp的所有调用均返回相同的值。
select current_timestamp(); -- 2021-11-18 22:24:17.432

-- 获取当前UNIX时间戳函数: unix_timestamp
select unix_timestamp(); -- 1637245506

-- UNIX时间戳转日期函数
```

```
select from_unixtime(1618238391); -- 2021-04-12 14:39:51
select from_unixtime(0, 'yyyy-MM-dd HH:mm:ss'); -- 1970-01-01 00:00:00

-- 日期转UNIX时间戳函数:
select unix_timestamp("2011-12-07 13:01:03"); -- 1323262863

-- 指定格式日期转UNIX时间戳函数: unix_timestamp
select unix_timestamp('20111207 13:01:03','yyyyMMdd HH:mm:ss');

-- 抽取日期函数: to_date
select to_date('2009-07-30 04:17:52'); -- 2009-07-30

-- 日期转年函数: year
select year('2009-07-30 04:17:52'); -- 2009

-- 日期转月函数: month
select month('2009-07-30 04:17:52'); -- 7

-- 日期转天函数: day
select day('2009-07-30 04:17:52');

-- 日期转小时函数: hour
select hour('2009-07-30 04:17:52');

-- 日期转分钟函数: minute
select minute('2009-07-30 04:17:52');

-- 日期转秒函数: second
select second('2009-07-30 04:17:52');

-- 日期转周函数: weekofyear 返回指定日期所示年份第几周
select weekofyear('2009-07-30 04:17:52');

-- 日期比较函数: datediff  日期格式要求'yyyy-MM-dd HH:mm:ss' or 'yyyy-MM-dd'
select datediff('2012-12-08','2012-12-10'); -- -2

-- 日期增加函数: date_add
select date_add('2012-02-28',10); -- 2012-03-09

-- 日期减少函数: date_sub
select date_sub('2012-01-1',10); -- 2011-12-22
```

3. 条件函数

主要用于条件判断、逻辑判断转换这样的场合:

返回值类型	函数名称	描述
T	if(boolean testCondition, T valueTrue, T valueFalseOrNull)	if条件判断
boolean	isnull(a) 、isnotnull (a)	空判断函数/非空判断函数
T	nvl(T value, T default_value)	空值转换函数
T	COALESCE(T v1, T v2, ...)	非空查找函数
T	CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END	条件转换函数
T	CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END	条件转换函数
T	nullif(a, b)	判断 a 与 b 是否相同， 相同返回 null ， 否则返回 a
void	assert_true(boolean condition)	如果'condition'不为真，则引发异常，否则返回null

示例:

```
-- if条件判断: if(boolean testCondition, T valueTrue, T valueFalseOrNull)
select if(1=2,100,200); -- 200
select if(sex ='男','M','W') from student limit 3;

-- 空判断函数: isnull( a )
select isnull("allen"); -- false
select isnull(null);
select isnull(sex) from xxx;

-- 非空判断函数: isnotnull ( a )
select isnotnull("allen");
select isnotnull(null);

-- 空值转换函数: nvl(T value, T default_value)
select nvl("allen","abc"); -- allen
select nvl(null,"abc"); -- abc

-- 非空查找函数: COALESCE(T v1, T v2, ...)
-- 返回参数中的第一个非空值；如果所有值都为NULL，那么返回NULL
select COALESCE(null,11,22,33);
```

```
select COALESCE(null,null,null,33);
select COALESCE(null,null,null);

-- 条件转换函数: CASE a WHEN b THEN c [WHEN d THEN e]* [ELSE f] END
select case 100
when 50 then 'tom'
when 100 then 'mary'
else 'tim' end;
-- mary
select case sex
when '男' then 'man'
else 'women' end
from student limit 3;

CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END
-- if else if else if  else
select case when 2 > 1 then 100
when 3 > 2 then 200
else 30 end; -- 100

-- assert_true(condition)
-- 如果'condition'不为真，则引发异常，否则返回null
SELECT assert_true(11 >= 0);
SELECT assert_true(-1 >= 0);
```

4. 字符串函数

主要针对字符串数据类型进行操作：

返回值类型	函数名称	描述
int	length(string A)	字符串长度函数
string	reverse(string A)	字符串反转函数
string	concat(string A, stringB...)	字符串连接函数
string	concat_ws(string SEP, string A, string B...)	带分隔符字符串连接函数
string	substr(string A, int start) substring(string A, int start)	字符串截取函数
string	upper(string A) ucase(string A)	字符串转大写函数
string	lower(string A) lcase(string A)	字符串转小写函数
string	trim(string A)	去空格函数
string	ltrim(string A)	左边去空格函数
string	rtrim(string A)	右边去空格函数
string	regexp_replace(string INITIAL_STRING, string PATTERN, string REPLACEMENT)	正则表达式替换函数
string	regexp_extract(string subject, string pattern, int index)	正则表达式解析函数
string	parse_url(string urlString, string partToExtract [, string keyToExtract])	URL解析函数
string	get_json_object(string json_string, string path)	json解析函数
string	space(int n)	空格字符串函数
string	repeat(string str, int n)	重复字符串函数
int	ascii(string str)	首字符ascii函数
string	lpad(string str, int len, string pad)	左补足函数
string	rpad(string str, int len, string pad)	右补足函数
array	split(string str, string pat)	分割字符串函数
int	find_in_set(string str, string strList)	集合查找函数

示例：

```
------------ String Functions 字符串函数- -----------
describe function extended find_in_set;

-- 字符串长度函数：
select length("abcd"); -- 4
```

```
-- 字符串反转函数：reverse
select reverse("abcd"); -- dcba

-- 字符串连接函数：concat(str1, str2, ... strN)
select concat("abcd","hello"); -- abcdhello

-- 带分隔符字符串连接函数：concat_ws(separator, [string | array(string)]+)
select concat_ws('.', 'www', array('baidu', 'com')); -- www.baidu.com

-- 字符串截取函数：substr(str, pos[, len]) 或者  substring(str, pos[, len])
select substr("helloworld",-2); -- ld -- pos是从1开始的索引，如果为负数则倒着数
select substr("helloworld",2,2); -- el

-- 字符串转大写函数：upper,ucase
select upper("helloworld");
select ucase("helloworld");

-- 字符串转小写函数：lower,lcase
select lower("HELLO");
select lcase("HELLO");

-- 去空格函数：trim 去除左右两边的空格
select trim(" helloworld ");

-- 左边去空格函数：ltrim
select ltrim(" helloworld ");

-- 右边去空格函数：rtrim
select rtrim(" helloworld ");

-- 正则表达式替换函数：regexp_replace(str, regexp, rep
select regexp_replace('100-200', '(\\d+)', 'num'); -- num-num

-- 正则表达式解析函数：regexp_extract(str, regexp[, idx]) 提取正则匹配到的指定组内容
select regexp_extract('100-200', '(\\d+)-(\\d+)', 2); -- 200

-- URL解析函数：parse_url 注意要想一次解析出多个 可以使用parse_url_tuple这个UDTF函数
select parse_url('http://www.baidu.cn/path/p1.php?query=1', 'HOST');

-- json解析函数：get_json_object
-- 空格字符串函数：space(n) 返回指定个数空格
select space(4);

-- 重复字符串函数：repeat(str, n) 重复str字符串n次
select repeat("ABC",2);

-- 首字符ascii函数：ascii
select ascii("abc");  --a对应ASCII 97

-- 左补足函数：lpad
select lpad('hi', 5, '??');  --???hi
select lpad('hi', 1, '??');  --h

-- 右补足函数：rpad
```

```
select rpad('hi', 5, '??');

-- 分割字符串函数: split(str, regex)
select split('apache hive', '\\s+'); -- ["apache","hive"]

-- 集合查找函数: find_in_set(str,str_array)
select find_in_set('a','abc,b,ab,c,def');
```

5. 类型转换函数

类型转换函数，主要用于显式的数据类型转换：

```
-- 任意数据类型之间转换:cast
select cast(12.14 as bigint);
select cast(12.14 as string);
```

6. 聚合函数

HQL提供了几种内置的UDAF聚合函数，例如max（...），min（...）和avg（...）。这些我们把它称之为基础的聚合函数。
通常情况下，聚合函数会与GROUP BY子句一起使用。 如果未指定GROUP BY子句，默认情况下，它会汇总所有行数据。

返回类型	语法	功能说明
BIGINT	count(*), count(expr), count(DISTINCT expr[, expr...])	count(*) 返回检索到的行总数，包括包含NULL值的行 count(col) 返回为其提供的列为非 NULL 的行数 count(DISTINCT col[, col]) 返回所提供的列唯一且非NULL的行数执行优化 hive.optimize.distinct.rewrite.
DOUBLE	sum(col), sum(DISTINCT col)	返回组中元素的总和或组中列的不同值的总和。
DOUBLE	avg(col), avg(DISTINCT col)	返回组中元素的平均值或组中列的不同值的平均值。
DOUBLE	min(col)	返回组中列的最小值。
DOUBLE	max(col)	返回组中列的最大值。
DOUBLE	variance(col), var_pop(col)	返回组中数字列的方差。
DOUBLE	var_samp(col)	返回组中数字列的无偏样本方差。

返回类型	语法	功能说明
DOUBLE	stddev_pop(col)	返回组中数字列的标准差。
DOUBLE	stddev_samp(col)	返回组中数字列的无偏样本标准差。
DOUBLE	covar_pop(col1, col2)	返回组中一对数字列的总体协方差。
DOUBLE	covar_samp(col1, col2)	返回组中一对数字列的样本协方差。
DOUBLE	corr(col1, col2)	返回组中一对数字列的皮尔逊相关系数。

创建示例表：

```
drop table if exists student;
create table student(
    num int,
    name string,
    sex string,
    age int,
    dept string)
row format delimited
fields terminated by ',';
```

加载数据：

```
load data local inpath '/root/data/stu.txt' into table student;
```

查询数据：

```
select * from student;
```

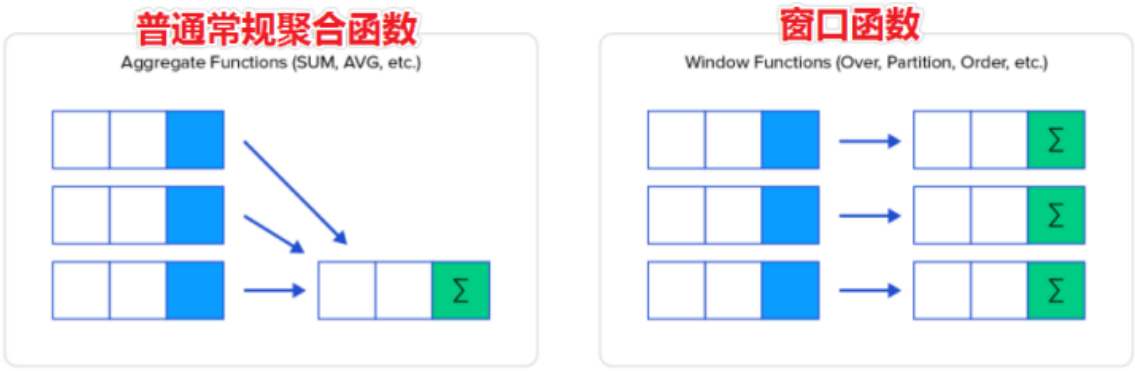
聚合函数示例：

```
-- 注意此处除了sex和聚合函数外不能再出现其他字段
select sex,count(*) as cnt from student group by sex;

-- 多个聚合函数一起使用
select count(*) as cnt1,avg(age) as cnt2 from student;

-- 聚合函数和case when条件转换函数、if函数使用
select
    sum(CASE WHEN sex = '男'THEN 1 ELSE 0 END)
from student;

select
    sum(if(sex = '男',1,0))
from student;

-- 聚合参数不支持嵌套聚合函数
select avg(count(*))  from student;
```

```
-- 聚合操作时针对null的处理

CREATE TABLE tmp_1 (val1 int, val2 int);
INSERT INTO TABLE tmp_1 VALUES (1, 2),(null,2),(2,3);

val1 val2
1,    2
null,  2
2,    3

select * from tmp_1;
select max(val1) from tmp_1; -- 2
select sum(val1) from tmp_1; -- 3 忽略了null

-- 第二行数据(NULL, 2) 在进行sum(val1 + val2)的时候会被忽略
select sum(val1 + val2) from tmp_1; -- 8

-- 解决方式:
select sum(case when val1 is null then 0 else val1 end + val2) from tmp_1; -- 10
select sum(coalesce(val1,0) + val2) from tmp_1; -- 10

-- 配合distinct关键字去重聚合
select count(distinct sex) as cnt1 from student;
select count(*) as gender_uni_cnt
from (select distinct sex from student) a;
```

7. 窗口函数

窗口函数（Window functions）是一种SQL函数，非常适合于数据分析，因此也叫做OLAP函数，其最大特点是：输入值是从SELECT语句的结果集中的一行或多行的“窗口”中获取的。你也可以理解为窗口有大有小（行有多有少）。
通过OVER子句，窗口函数与其他SQL函数有所区别。如果函数具有OVER子句，则它是窗口函数。如果它缺少OVER子句，则它是一个普通的聚合函数。
窗口函数可以简单地解释为类似于聚合函数的计算函数，但是通过GROUP BY子句组合的常规聚合会隐藏正在聚合的各个行，最终输出一行，窗口函数聚合后还可以访问当中的各个行，并且可以将这些行中的某些属性添加到结果集中。



语法：

```
Function(arg1,..., argn) OVER ([PARTITION BY <...>] [ORDER BY <....>]
[<window_expression>])

-- 其中Function(arg1,..., argn) 可以是下面分类中的任意一个
    -- 聚合函数：比如sum max avg等
    -- 排序函数：比如rank row_number等
    -- 分析函数：比如lead lag first_value等

-- OVER [PARTITION BY <...>] 类似于group by 用于指定分组   每个分组你可以把它叫做窗口
-- 如果没有PARTITION BY 那么整张表的所有行就是一组

-- [ORDER BY <....>]  用于指定每个分组内的数据排序规则 支持ASC、DESC

-- [<window_expression>] 用于指定每个窗口中 操作的数据范围 默认是窗口中所有行
```

附录

Hive文档：https://cwiki.apache.org/confluence/display/Hive/LanguageManual