表连接就是通过关联多张表，从而检索出需要的数据的方法

实际的项目，存在多张表的关联关系。不可能在一张表里面就能检索出所有数据。如果没有表连接的话，那么我们就需要非常多的操作。比如需要从A表找出限制性的条件来从B表中检索数据。不但需要分多表来操作，而且效率也不高。

下面我们详细来看看表连接。表连接有多种不同的类型，有交叉连接（CROSS JOIN）、内连接（INNER JOIN）、外连接（OUTTER JOIN）。

（1）内连接（INNER JOIN）：内连接组合两张表，并且只获取满足两表连接条件的数据。

**1、内联接**（典型的联接运算，使用像 =  或 <> 之类的比较运算符）。包括相等联接和自然联接。       
内联接使用比较运算符根据每个表共有的列的值匹配两个表中的行。例如，检索 students和courses表中学生标识号相同的所有行。     
**2、外联接。**外联接可以是左向外联接、右向外联接或完整外部联接。       
在 FROM子句中指定外联接时，可以由下列几组关键字中的一组指定：       
1）LEFT  JOIN或LEFT OUTER JOIN       
左向外联接的结果集包括  LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。         
2）RIGHT  JOIN 或 RIGHT  OUTER  JOIN       
右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。         
3）FULL  JOIN 或 FULL OUTER JOIN  
完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。     
  
**3、交叉联接**交叉联接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉联接也称作笛卡尔积。      
FROM 子句中的表或视图可通过内联接或完整外部联接按任意顺序指定；但是，用左或右向外联接指定表或视图时，表或视图的顺序很重要。有关使用左或右向外联接排列表的更多信息，请参见使用外联接。       
  
**例子：**     
-------------------------------------------------  
  a表     id   name     b表     id   job   parent\_id     
              1   张3                   1     23     1     
              2   李四                 2     34     2     
              3   王武                 3     34     4         
  a.id同parent\_id   存在关系     
--------------------------------------------------      
 1） 内连接     
  select   a.\*,b.\*   from   a   inner   join   b     on   a.id=b.parent\_id         
  结果是       
  1   张3                   1     23     1     
  2   李四                  2     34     2     
  
  2）左连接     
  select   a.\*,b.\*   from   a   left   join   b     on   a.id=b.parent\_id         
  结果是       
  1   张3                   1     23     1     
  2   李四                  2     34     2     
  3   王武                  null     
  
 3） 右连接     
  select   a.\*,b.\*   from   a   right   join   b     on   a.id=b.parent\_id         
  结果是       
  1   张3                   1     23     1     
  2   李四                  2     34     2     
  null                       3     34     4     
  
 4） 完全连接     
  select   a.\*,b.\*   from   a   full   join   b     on   a.id=b.parent\_id     
  结果是       
  1   张3                  1     23     1     
  2   李四                 2     34     2     
  null               　　  3     34     4     
  3   王武                 null  
--------------------------------------------------------------------------------------------**一、交叉连接（CROSS JOIN）**交叉连接（CROSS JOIN）：有两种，显式的和隐式的，不带ON子句，返回的是两表的乘积，也叫笛卡尔积。  
例如：下面的语句1和语句2的结果是相同的。

**语句1：隐式的交叉连接，没有CROSS JOIN。**SELECT O.ID, O.ORDER\_NUMBER, C.ID, C.NAME  
FROM ORDERS O , CUSTOMERS C  
WHERE O.ID=1;

**语句2：显式的交叉连接，使用CROSS JOIN。**SELECT O.ID,O.ORDER\_NUMBER,C.ID,  
C.NAME  
FROM ORDERS O CROSS JOIN CUSTOMERS C  
WHERE O.ID=1;  
语句1和语句2的结果是相同的，查询结果如下：

**二、内连接（INNER JOIN）**内连接（INNER JOIN）：有两种，显式的和隐式的，返回连接表中符合连接条件和查询条件的数据行。（所谓的链接表就是数据库在做查询形成的中间表）。  
例如：下面的语句3和语句4的结果是相同的。

**语句3：隐式的内连接，没有INNER JOIN，形成的中间表为两个表的笛卡尔积。**SELECT O.ID,O.ORDER\_NUMBER,C.ID,C.NAME  
FROM CUSTOMERS C,ORDERS O  
WHERE C.ID=O.CUSTOMER\_ID;

**语句4：显示的内连接，一般称为内连接，有INNER JOIN，形成的中间表为两个表经过ON条件过滤后的笛卡尔积。**SELECT O.ID,O.ORDER\_NUMBER,C.ID,C.NAME  
FROM CUSTOMERS C INNER JOIN ORDERS O ON C.ID=O.CUSTOMER\_ID;  
语句3和语句4的查询结果：

**三、外连接（OUTER JOIN）：**外连不但返回符合连接和查询条件的数据行，还返回不符合条件的一些行。外连接分三类：左外连接（LEFT OUTER JOIN）、右外连接（RIGHT OUTER JOIN）和全外连接（FULL OUTER JOIN）。  
三者的共同点是都返回符合连接条件和查询条件（即：内连接）的数据行。不同点如下：  
左外连接还返回左表中不符合连接条件单符合查询条件的数据行。  
右外连接还返回右表中不符合连接条件单符合查询条件的数据行。  
全外连接还返回左表中不符合连接条件单符合查询条件的数据行，并且还返回右表中不符合连接条件单符合查询条件的数据行。全外连接实际是上左外连接和右外连接的数学合集（去掉重复），即“全外=左外 UNION 右外”。  
说明：左表就是在“（LEFT OUTER JOIN）”关键字左边的表。右表当然就是右边的了。在三种类型的外连接中，OUTER 关键字是可省略的。

下面举例说明：  
**语句5：左外连接（LEFT OUTER JOIN）**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O LEFT OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID;

**语句6：右外连接（RIGHT OUTER JOIN）**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O RIGHT OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID;  
注意：WHERE条件放在ON后面查询的结果是不一样的。例如：

**语句7：WHERE条件独立。**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O LEFT OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID  
WHERE O.ORDER\_NUMBER<>'MIKE\_ORDER001';

**语句8：将语句7中的WHERE条件放到ON后面。**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O LEFT OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID AND O.ORDER\_NUMBER<>'MIKE\_ORDER001';

从语句7和语句8查询的结果来看，显然是不相同的，语句8显示的结果是难以理解的。因此，推荐在写连接查询的时候，ON后面只跟连接条件，而对中间表限制的条件都写到WHERE子句中。  
  
**语句9：全外连接（FULL OUTER JOIN）。**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O FULL OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID;  
注意：MySQL是不支持全外的连接的，这里给出的写法适合Oracle和DB2。但是可以通过左外和右外求合集来获取全外连接的查询结果。下图是上面SQL在Oracle下执行的结果：  
  
**语句10：左外和右外的合集，实际上查询结果和语句9是相同的。**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O LEFT OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID  
UNION  
SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O RIGHT OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID;  
语句9和语句10的查询结果是相同的，如下：

**四、联合连接（UNION JOIN）：**这是一种很少见的连接方式。Oracle、MySQL均不支持，其作用是：找出全外连接和内连接之间差异的所有行。这在数据分析中排错中比较常用。也可以利用数据库的集合操作来实现此功能。  
**语句11：联合查询（UNION JOIN）例句，还没有找到能执行的SQL环境。**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O UNION JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID  
  
**语句12：语句11在DB2下的等价实现。还不知道DB2是否支持语句11呢！**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O FULL OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID  
EXCEPT  
SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O INNER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID;  
  
**语句13：语句11在Oracle下的等价实现。**SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O FULL OUTER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID  
MINUS  
SELECT O.ID,O.ORDER\_NUMBER,O.CUSTOMER\_ID,C.ID,C.NAME  
FROM ORDERS O INNER JOIN CUSTOMERS C ON C.ID=O.CUSTOMER\_ID;  
**查询结果如下：**

**五、自然连接（NATURAL INNER JOIN）：**说真的，这种连接查询没有存在的价值，既然是SQL2标准中定义的，就给出个例子看看吧。自然连接无需指定连接列，SQL会检查两个表中是否相同名称的列，且假设他们在连接条件中使用，并且在连接条件中仅包含一个连接列。不允许使用ON语句，不允许指定显示列，显示列只能用\*表示（ORACLE环境下测试的）。对于每种连接类型（除了交叉连接外），均可指定NATURAL。下面给出几个例子。  
**语句14：**SELECT \*  
FROM ORDERS O NATURAL INNER JOIN CUSTOMERS C;

**语句15：**SELECT \*  
FROM ORDERS O NATURAL LEFT OUTER JOIN CUSTOMERS C;

**语句16：**SELECT \*  
FROM ORDERS O NATURAL RIGHT OUTER JOIN CUSTOMERS C;

**语句17：**SELECT \*  
FROM ORDERS O NATURAL FULL OUTER JOIN CUSTOMERS C;

**六、SQL查询的基本原理：两种情况介绍。  
第一、**单表查询：根据WHERE条件过滤表中的记录，形成中间表（这个中间表对用户是不可见的）；然后根据SELECT的选择列选择相应的列进行返回最终结果。  
  
**第二、**两表连接查询：对两表求积（笛卡尔积）并用ON条件和连接连接类型进行过滤形成中间表；然后根据WHERE条件过滤中间表的记录，并根据SELECT指定的列返回查询结果。  
 **第三、**多表连接查询：先对第一个和第二个表按照两表连接做查询，然后用查询结果和第三个表做连接查询，以此类推，直到所有的表都连接上为止，最终形成一个中间的结果表，然后根据WHERE条件过滤中间表的记录，并根据SELECT指定的列返回查询结果。  
理解SQL查询的过程是进行SQL优化的理论依据。

**七、ON后面的条件（ON条件）和WHERE条件的区别：**ON条件：是过滤两个链接表笛卡尔积形成中间表的约束条件。  
WHERE条件：在有ON条件的SELECT语句中是过滤中间表的约束条件。在没有ON的单表查询中，是限制物理表或者中间查询结果返回记录的约束。在两表或多表连接中是限制连接形成最终中间表的返回结果的约束。  
从这里可以看出，将WHERE条件移入ON后面是不恰当的。推荐的做法是：  
ON只进行连接操作，WHERE只过滤中间表的记录。

**八、总结**连接查询是SQL查询的核心，连接查询的连接类型选择依据实际需求。如果选择不当，非但不能提高查询效率，反而会带来一些逻辑错误或者性能低下。下面总结一下两表连接查询选择方式的依据：  
1、 查两表关联列相等的数据用内连接。  
2、 Col\_L是Col\_R的子集时用右外连接。  
3、 Col\_R是Col\_L的子集时用左外连接。  
4、 Col\_R和Col\_L彼此有交集但彼此互不为子集时候用全外。  
5、 求差操作的时候用联合查询。  
多个表查询的时候，这些不同的连接类型可以写到一块。例如：  
SELECT T1.C1,T2.CX,T3.CY  
FROM TAB1 T1  
       INNER JOIN TAB2 T2 ON (T1.C1=T2.C2)  
       INNER JOIN TAB3 T3 ON (T1.C1=T2.C3)  
       LEFT OUTER JOIN TAB4 ON(T2.C2=T3.C3);  
WHERE T1.X >T3.Y;

Hive操作语句的使用实例讲解

#创建表人信息表 person(String name,int age)

hive> create table person(name STRING,age INT)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ESCAPED BY '\\' STORED AS TEXTFILE;

OK

Time taken: 0.541 seconds

#创建表票价信息表 ticket(int age,float price)

hive> create table ticket(age INT,price FLOAT)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ESCAPED BY '\\' STORED AS TEXTFILE;

OK

Time taken: 0.154 seconds

#创建本地数据文件

-rw-rw-r-- 1 hadoop hadoop 40 Feb 6 13:28 person.txt

-rw-rw-r-- 1 hadoop hadoop 45 Feb 6 13:28 ticket.txt

#将本地的数据文件load到hive数据仓库中

hive> LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/person.txt' OVERWRITE INTO TABLE person;

Copying data from file:/home/hadoop/hfxdoc/person.txt

Copying file: file:/home/hadoop/hfxdoc/person.txt

Loading data to table default.person

Deleted hdfs://10.15.107.155:8000/user/hive/warehouse/person

OK

Time taken: 0.419 seconds

hive> LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/ticket.txt' OVERWRITE INTO TABLE ticket;

Copying data from file:/home/hadoop/hfxdoc/ticket.txt

Copying file: file:/home/hadoop/hfxdoc/ticket.txt

Loading data to table default.ticket

Deleted hdfs://10.15.107.155:8000/user/hive/warehouse/ticket

OK

Time taken: 0.25 seconds

#load命令会将数据文件移动到配置好的数据路径下：/user/hive/warehouse

hive> show tables;

hive> describe person

hive> select \* from person;

OK

huang 26

lili 25

dongdong 13

wangxiao 5

Time taken: 0.092 seconds

hive>

#注意select \*语句是不会编译成MapReduce程序的，所以很快。

#稍作复杂点的join查询

hive> select \* from person join ticket on person.age = ticket.age;

MapReduce Total cumulative CPU time: 5 seconds 510 msec

Ended Job = job\_201301211420\_0011

MapReduce Jobs Launched:

Job 0: Map: 2 Reduce: 1 Cumulative CPU: 5.51 sec HDFS Read: 519 HDFS Write: 71 SUCCESS

Total MapReduce CPU Time Spent: 5 seconds 510 msec

OK

wangxiao 5 5 10.0

dongdong 13 13 20.0

lili 25 25 30.0

huang 26 26 30.0

Time taken: 32.465 seconds

#这里查询语句被编译成MapReduce程序，在hadoop上执行

#采用外部表

#首先将本地文件put到hdfs文件路径下

[hadoop@localhost hfxdoc]$ hadoop fs -mkdir /tmp/ticket

[hadoop@localhost hfxdoc]$ hadoop fs -put person.txt /tmp/ticket

[hadoop@localhost hfxdoc]$ hadoop fs -put ticket.txt /tmp/ticket

[hadoop@localhost hfxdoc]$ hadoop fs -ls /tmp/ticket

Found 2 items

-rw-r--r-- 1 hadoop supergroup 40 2013-02-06 13:45 /tmp/ticket/person.txt

-rw-r--r-- 1 hadoop supergroup 45 2013-02-06 13:45 /tmp/ticket/ticket.txt

create external table person\_ext(name STRING,age INT)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ESCAPED BY '\\' STORED AS TEXTFILE LOCATION '/tmp/ticket'

#LOCATION只能配置数据路径，而刚刚我们的路径下有两个表的文件？这样创建的其中一个表可以吗？

#不可以！所以，一个文件路径下面的所有文件都应该是关联这个数据表的数据文件。

#如果有其他表的文件，这个创建过程不会报错，因为，hive默认文本里的字符串类型都可以隐式转换成任何其他数据类型。比如你还有一个文件是一行三列的，那么第三列

#在person表中是解析不到的，如果每行只有一列，那么第二列将会用NULL来补齐。所以我们调整下hdfs文件路径。

hive> select \* from person\_ext;

OK

huang 26

lili 25

dongdong 13

wangxiao 5

1 10

2 10

5 10

13 20

14 20

25 30

26 30

31 40

Time taken: 0.088 seconds

hive> drop table person\_ext;

#Drop外表的操作不会删除元信息以为的数据，所以hdfs上还是存在数据文件

#复杂类型的数据表，这里列之间以'\t'分割，数组元素之间以','分割

#数据文件内容如下

1 huangfengxiao beijing,shanghai,tianjin,hangzhou

2 linan changchu,chengdu,wuhan

hive> create table complex(name string,work\_locations array<string>)

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '\t'

> COLLECTION ITEMS TERMINATED BY ',';

hive> describe complex;

OK

name string

work\_locations array<string>

hive> LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/complex.txt' OVERWRITE INTO TABLE complex

hive> select \* from complex;

OK

huangfengxiao ["beijing","shanghai","tianjin","hangzhou"]

linan ["changchu","chengdu","wuhan"]

Time taken: 0.125 seconds

hive> select name, work\_locations[0] from complex;

MapReduce Total cumulative CPU time: 790 msec

Ended Job = job\_201301211420\_0012

MapReduce Jobs Launched:

Job 0: Map: 1 Cumulative CPU: 0.79 sec HDFS Read: 296 HDFS Write: 37 SUCCESS

Total MapReduce CPU Time Spent: 790 msec

OK

huangfengxiao beijing

linan changchu

Time taken: 20.703 seconds

#如何分区？

表class(teacher sting,student string,age int)

Mis li huangfengxiao 20

Mis li lijie 21

Mis li dongdong 21

Mis li liqiang 21

Mis li hemeng 21

Mr xu dingding 19

Mr xu wangqiang 19

Mr xu lidong 19

Mr xu hexing 19

如果我们将这个班级成员的数据按teacher来分区

create table classmem(student string,age int) partitioned by(teacher string)

分区文件

classmem\_Misli.txt

huangfengxiao 20

lijie 21

dongdong 21

liqiang 21

hemeng 21

classmem\_MrXu.txt

dingding 19

wangqiang 19

lidong 19

hexing 19

LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/classmem\_Misli.txt' INTO TABLE classmem partition (teacher = 'Mis.li')

LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/classmem\_MrXu.txt' INTO TABLE classmem partition (teacher = 'Mis.Xu')

#分区列被默认到最后一列

hive> select \* from classmem where teacher = 'Mr.Xu';

OK

dingding 19 NULL Mr.Xu

wangqiang 19 NULL Mr.Xu

lidong 19 NULL Mr.Xu

hexing 19 NULL Mr.Xu

Time taken: 0.196 seconds

#直接从分区检索，加速；如果where子句的条件不是分区列，那么，这个sql将被编译成mapreduce程序，延时很大。

#所以，我们建立分区，是为了一些常用的筛选查询字段而用的。

#桶的使用？更高效！可取样！主要用于大数据集的取样

桶的原理是对一个表（或者分区）进行切片，选择被切片的字段，设定桶的个数，用字段与个数的hash值进行入桶。

比如bucket.txt数据文件内容如下：

id name age

1 huang 11

2 li 11

3 xu 12

4 zhong 14

5 hu 15

6 liqiang 17

7 zhonghua 19

如果我们想将这个数据表切成3个桶，切片字段为id

那么用id字段hash后，3个桶的内容如下：

桶id hash 3 =0

3 xu 12

6 liqiang 17

桶id hash 3 =1

1 huang 11

4 zhong 14

7 zhonghua 19

桶id hash 3 =2

2 li 11

5 hu 15

这个过程的创建表语句如下：

create table bucketmem (id int,name string,age int) CLUSTERED BY (id) sorted by (id asc) into 3 buckets

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/home/hadoop/hfxdoc/bucketmem.txt' INTO TABLE bucketmem;

select \* from bucketmem tablesample(bucket 1 out of 4)

#其他操作参考，更完整的请参考官网： https://cwiki.apache.org/confluence/display/Hive/Home

1) 创建与已知表相同结构的表Like：

只复制表的结构，而不复制表的内容。

create table test\_like\_table like test\_bucket;

2) 对表进行重命名 rename to：

ALTER TABLE table\_name RENAME TO new\_table\_name

3) 增加分区 Add Partitions：

ALTER TABLE table\_name ADD partition\_spec [ LOCATION 'location1' ]partition\_spec [ LOCATION 'location2' ]

4) 对表中的某一列进行修改，包括列的名称/列的数据类型/列的位置/列的注释

ALTER TABLE table\_name CHANGE [COLUMN] col\_old\_name col\_new\_name column\_type[COMMENT col\_comment] [FIRST|AFTER column\_name]

5) 添加/替换列Add/ReplaceColumns

ALTER TABLE table\_name ADD|REPLACE COLUMNS (col\_name data\_type [COMMENTcol\_comment], ...)

ADD COLUMNS 允许用户在当前列的末尾增加新的列，但是在分区列之前。

6) 创建表的完整语句：

Create [EXTERNAL] TABLE [IF NOT EXISTS] table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[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]

[ROW FORMAT row\_format]

[STORED AS file\_format]

[LOCATION hdfs\_path]

7) 在hive中查看hdfs文件

>dfs -ls /user;