链滴

4-DQL 语句

作者：Carey

原文链接：https://ld246.com/article/1614771000613

来源网站：链滴

3.7 DQL 语句

3.7.1 单表操作

语法：

```
SELECT
[ALL | DISTINCT | DISTINCTROW ]
[SQL_CACHE | SQL_NO_CACHE]
select_expr [, select_expr ...]
[FROM table_references
[WHERE where_condition]
[GROUP BY {col_name | expr | position}
[ASC | DESC], ... [WITH ROLLUP]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position}
[ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[FOR UPDATE | LOCK IN SHARE MODE]
```

说明：

- 字段显示可以使用别名：
  - col1 AS alias1, col2 AS alias2, ...
- WHERE子句：指明过滤条件以实现"选择"的功能：
  - 过滤条件：布尔型表达式
- 算术操作符：+, -, *, /, %

- 比较操作符：=,<=>（相等或都为空），<>, !=(非标准SQL), >, >=, <, <=
- BETWEEN min_num AND max_num，一般用于去范围条件
- IN (element1, element2, ...)，一般用于取几个条件也就是或者
- IS NULL, IS NOT NULL，判断空值
- DISTINCT 去除重复行，范例：SELECT DISTINCT gender FROM students;
- LIKE: % 任意长度的任意字符 _ 任意单个字符
- RLIKE：正则表达式，索引失效，不建议使用
- REGEXP：匹配字符串可用正则表达式书写模式，同上
- 逻辑操作符：NOT，AND，OR，XOR
- GROUP：根据指定的条件把查询结果进行"分组"以用于做"聚合"运算
  - 常见聚合函数：avg(), max(), min(), count(), sum()
  - HAVING: 对分组聚合运算后的结果指定过滤条件
  - 一旦分组group by ，select语句后只跟分组的字段，聚合函数
- ORDER BY: 根据指定的字段对查询结果进行排序
  - 升序：ASC
  - 降序：DESC
- LIMIT [[offset,]row_count]：对查询的结果进行输出行数数量限制
- 对查询结果中的数据请求施加"锁
  - FOR UPDATE: 写锁，独占或排它锁，只有一个读和写操作
  - LOCK IN SHARE MODE: 读锁，共享锁，同时多个读操作

范例：密码生成

```
mariadb root@(none):(none)> select password("zhangzhuo");
+-------------------------------------------+
| password("zhangzhuo")                     |
+-------------------------------------------+
| *E537F5F82C1F36D566632B4C9061BD6715BABF7C |
+-------------------------------------------+
1 row in set
Time: 0.013s
```

范例：简单查询

```
#查看表结构
mysql root@(none):hellodb> desc students
#插入数据
mysql root@(none):hellodb> insert into students values(1,'tom','m'),(2,'alice','f');
mysql root@(none):hellodb> select * from students where `StuID` <3;
mysql root@(none):hellodb> select * from students where `Gender`='m';
#查询字段空值或非空值
mysql root@(none):hellodb> select * from students where `ClassID` is null;
mysql root@(none):hellodb> select * from students where `ClassID` is not null;
```

```
#分组后降序排序查询前2行
mysql root@(none):hellodb> select * from students order by name desc limit 2;
#分组后降序排序查询跳过第一行查询后2行
mysql root@(none):hellodb> select * from students order by name desc limit 1,2;
#范围查询俩种方式
mysql root@(none):hellodb> select * from students where `StuID` >=2 and `StuID` <=4;
mysql root@(none):hellodb> select * from students where  `StuID` between 2 and 4;
#模糊查询
mysql root@(none):hellodb> select * from students where name like 't%';
#正则表达式查询
mysql root@(none):hellodb> select * from students where name rlike '.*[lo].*';
#查询结果命名别名
mysql root@(none):hellodb> select `StuID` id,name as stuname from students;
#包含查询
mysql root@(none):hellodb> select * from students where `ClassID` in (1,3,5);
mysql root@(none):hellodb> select * from students where `ClassID` not in (1,3,5);
```

范例：判断是否为NULL俩种方式或者不为NULL

```
mysql root@(none):hellodb> select * from students where `ClassID` is null;
+-------+-------------+-----+--------+---------+-----------+
| StuID | Name        | Age | Gender | ClassID | TeacherID |
+-------+-------------+-----+--------+---------+-----------+
| 24    | Xu Xian     | 27  | M      |    |     |
| 25    | Sun Dasheng | 100 | M      |    |     |
+-------+-------------+-----+--------+---------+-----------+

2 rows in set
Time: 0.007s
mysql root@(none):hellodb> select * from students where `ClassID` <=> null;
+-------+-------------+-----+--------+---------+-----------+
| StuID | Name        | Age | Gender | ClassID | TeacherID |
+-------+-------------+-----+--------+---------+-----------+
| 24    | Xu Xian     | 27  | M      |    |     |
| 25    | Sun Dasheng | 100 | M      |    |     |
+-------+-------------+-----+--------+---------+-----------+

2 rows in set
Time: 0.007s
#不为空的
mysql root@(none):hellodb> select * from students where `ClassID` is not null;
```

范例：去重

```
mysql root@(none):hellodb> select distinct gender from students;
+--------+
| gender |
+--------+
| M      |
| F      |
+--------+
2 rows in set
Time: 0.008s
```

范例：分组统计

```
#查看每个班级中人数
mysql root@(none):hellodb> select `ClassID`,count(*) from students group by classid;
+---------+----------+
| ClassID | count(*) |
+---------+----------+
| 2       | 3        |
| 1       | 4        |
| 4       | 4        |
| 3       | 4        |
| 5       | 1        |
| 7       | 3        |
| 6       | 4        |
|   | 2        |
+---------+----------+
8 rows in set
Time: 1.634s
#查看每个班级中男生女生分别的人数

mysql root@(none):hellodb> select `ClassID`,`Gender`,count(*) as 数量 from students group by
classid,ge
-> nder
+---------+--------+------+
| ClassID | Gender | 数量 |
+---------+--------+------+
| 2       | M      | 3    |
| 1       | M      | 2    |
| 4       | M      | 4    |
| 3       | M      | 1    |
| 5       | M      | 1    |
| 3       | F      | 3    |
| 7       | F      | 2    |
| 6       | F      | 3    |
| 6       | M      | 1    |
| 1       | F      | 2    |
| 7       | M      | 1    |
|   | M      | 2    |
+---------+--------+------+
12 rows in set
Time: 0.007s
#统计班级编号大于3的且班级中平均年纪大于30的班级
mysql root@(none):hellodb> select classid,avg(age) as 平均年龄 from students where `ClassID`
>3 group b
-> y classid having 平均年龄 >30;
+---------+----------+
| classid | 平均年龄 |
+---------+----------+
| 5       | 46.0000  |
+---------+----------+
1 row in set
Time: 0.008s
#统计学生表中男生的平均年龄
mysql root@(none):hellodb> select gender,avg(age) 平均年龄 from students group by gender
having `Gender -> `='M';
#多个字段分组统计，分组在前在后不影响结果
```

```
mysql root@(none):hellodb> select classid,gender,count(*) 数量 from students group by class
d,gender;
mysql root@(none):hellodb> select classid,gender,count(*) 数量 from students group by gen
er,classid;
#注意:一旦使用分组group by,在select 后面的只能采用分组的列和聚合函数,其它的列不能放在selec
后面,否则根据系统变量SQL_MODE的值不同而不同的结果
```

范例：排序

```
#年龄按降序排序取前三个
mysql root@(none):hellodb> select * from students order by `Age` desc limit 3;
#年龄按降序排序跳过前三个取后续的2个
mysql root@(none):hellodb> select * from students order by `Age` desc limit 3,2;
#按班级id字段非空分组后年龄求和后按班级id升序排序，分组前排除班级id字段是null的
mysql root@(none):hellodb> select classid,sum(age) from students where `ClassID` is not null
group by c
-> lassid order by `ClassID`;
#按班级id字段非空分组后年龄求和后按班级id升序排序，分组后排除班级id字段是null的
mysql root@(none):hellodb> select classid,sum(age) from students group by classid having cl
ssid is not
->  null order by `ClassID`;
#按班级id字段非空分组后年龄求和后按班级id升序排序跳过前2个取后续的3个
mysql root@(none):hellodb> select classid,sum(age) from students where `ClassID` is not null
group by c
-> lassid order by `ClassID` limit 2,3;
#过滤班级id是null的按性别降序排序后年龄按升序排序
mysql root@(none):hellodb> select * from students where `ClassID` is not null order by `Gend
r` desc,ag
-> e asc;
#多列排序，按性别降序排序后年龄按升序排序
mysql root@(none):hellodb> select * from students order by `Gender` desc,age asc;
```

范例：分组和排序

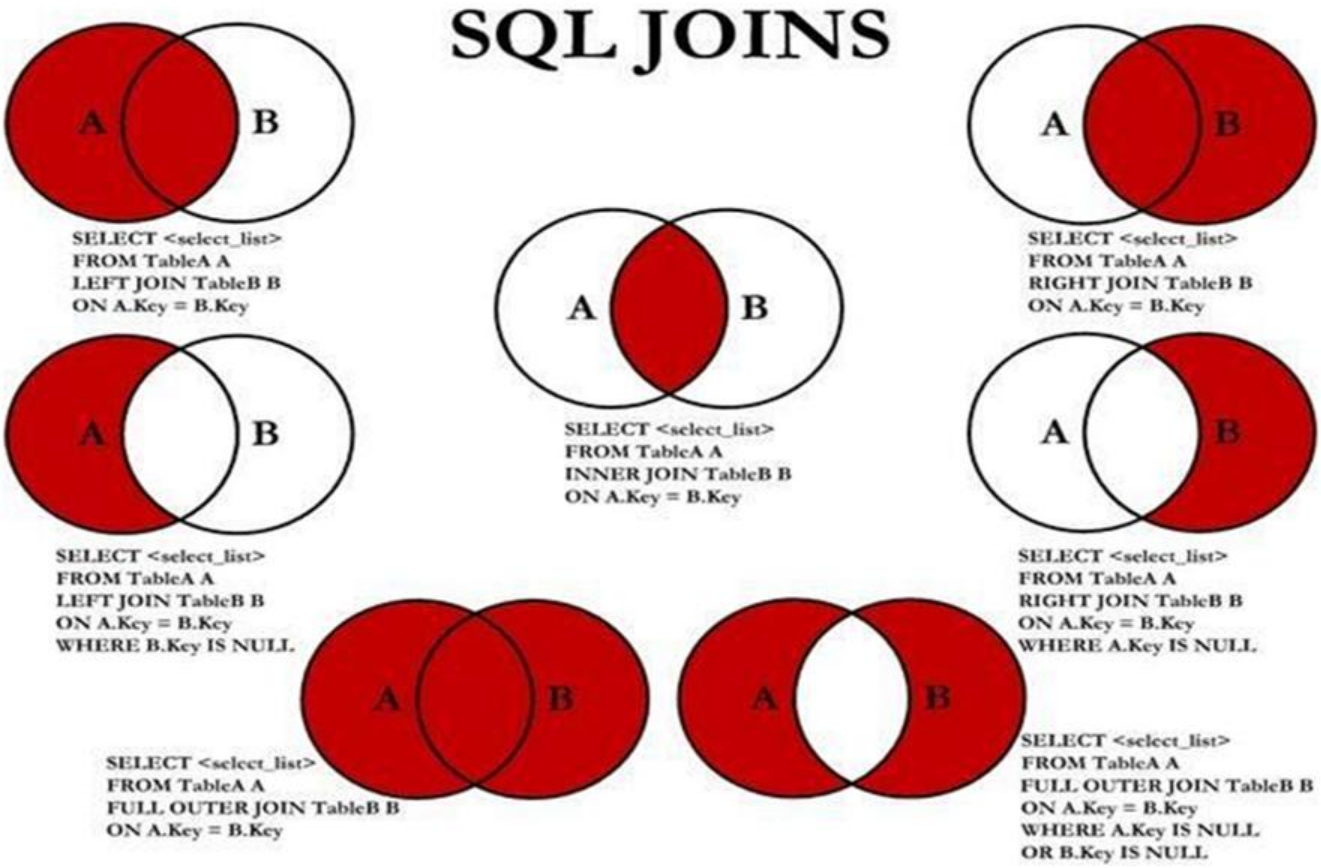
```
#按班级id分组，统计各个班级数量后，数量按升序排序
mysql root@(none):hellodb> select classid,count(*) 数量 from students group by classid order
by 数量;
#以性别和班级分组排除班级id是空的，求他们的平均年龄后，按照性别先升序排序在按照班级id升
排序
mysql root@(none):hellodb> select gender,classid,avg(age) from students where `ClassID` is
ot null gro
-> up by gender,classid order by `Gender`,`ClassID`;
#按照年龄升序排序取前10个
mysql root@(none):hellodb> select * from students order by `Age` limit 10;
#按照年龄升序排序跳过前3个取后续的10个
mysql root@(none):hellodb> select * from students order by `Age` limit 3,10;
#先对年龄去重后年龄按照升序排序后取前3个
mysql root@(none):hellodb> select distinct age from students order by `Age` limit 3;
#先对年龄去重后年龄按照升序排序后跳过前3个取后续的5个
mysql root@(none):hellodb> select distinct age from students order by age limit 3,5;

#分组和排序的次序group by,having,order by
#以下顺序会出错,group by,order by,having
#以下顺序会出错,order by,group by,having
```

范例：时间字段进行过滤查询,并且timestamp可以随其它字段的更新自动更新

```
mysql root@(none):hellodb> create table testdata (id int auto_increment primary key,date ti
estamp defa
-> ult current_timestamp on update current_timestamp );
mysql root@(none):hellodb> insert into testdata () values(),(),();
mysql root@(none):hellodb> select * from testdata;
+----+---------------------+
| id | date                |
+----+---------------------+
| 1  | 2021-02-04 14:47:34 |
| 2  | 2021-02-04 14:47:34 |
| 3  | 2021-02-04 14:47:34 |
+----+---------------------+
mysql root@(none):hellodb> select * from testdata where `date` between '2021-02-04 14:40:
0' and '2021-
-> 02-04 14:50:00';
+----+---------------------+
| id | date                |
+----+---------------------+
| 1  | 2021-02-04 14:47:34 |
| 2  | 2021-02-04 14:47:34 |
| 3  | 2021-02-04 14:47:34 |
+----+---------------------+
#修改其它字段,会自动更新timestamp字段
mysql root@(none):hellodb> update testdata set id=10 where id=1;
Query OK, 1 row affected
Time: 0.002s
mysql root@(none):hellodb> select * from testdata;
+----+---------------------+
| id | date                |
+----+---------------------+
| 2  | 2021-02-04 14:52:34 |
| 3  | 2021-02-04 14:52:34 |
| 4  | 2021-02-04 14:52:38 |
| 5  | 2021-02-04 14:52:39 |
| 10 | 2021-02-04 14:53:07 |
+----+---------------------+
5 rows in set
Time: 0.006s
```

3.7.2 多表查询

多表查询，即查询结果来自于多张表



- 子查询：在SQL语句嵌套着查询语句，性能较差，基于某语句的查询结果再次进行的查询
- 联合查询：UNION
- 交叉连接：笛卡尔乘积 CROSS JOIN
- 内连接：
  - 等值连接：让表之间的字段以"等值"建立连接关系
  - 不等值连接
  - 自然连接：去掉重复列的等值连接 , 语法: FROM table1 NATURAL JOIN table2;
- 外连接：
  - 左外连接：FROM tb1 LEFT JOIN tb2 ON tb1.col=tb2.col
  - 右外连接：FROM tb1 RIGHT JOIN tb2 ON tb1.col=tb2.col
  - 完全外连接: FROM tb1 FULL OUTER JOIN tb2 ON tb1.col=tb2.col 注意:MySQL 不支持此SQ语法
- 自连接：本表和本表进行连接查询

3.7.2.1 子查询

子查询 subquery 即SQL语句调用另一个SELECT子句,可以是对同一张表,也可以是对不同表,主要有以四种常见的用法：

1. 用于比较表达式中的子查询；子查询仅能返回单个值

```
#查询学生表中年龄大于老师表中平均年龄
mysql root@(none):hellodb> select name,age from students where `Age`>(select avg(`Age`) f
om teachers);
```

2. 用于IN中的子查询：子查询应该单独查询并返回一个或多个值重新构成列表

```
#查询学生表中年龄和老师表中年龄一致的
mysql root@(none):hellodb> select name,age from students where age in(select age from tea
hers);
```

3. 用于EXISTS 和 Not EXISTS

参考链接：https://dev.mysql.com/doc/refman/8.0/en/exists-and-not-exists-subqueries.html

EXISTS(包括 NOT EXISTS)子句的返回值是一个BOOL值。 EXISTS 内部有一个子查询语句(SELECT...ROM...)， 将其称为EXIST的内查询语句。其内查询语句返回一个结果集。 EXISTS子句根据其内查询句的结果集空或者非空，返回一个布尔值。将外查询表的每一行，代入内查询作为检验，如果内查询回的结果为非空值，则EXISTS子句返回TRUE，外查询的这一行数据便可作为外查询的结果行返回，则不能作为结果

```
#查询学生表中有对应老师的学生
mysql root@(none):hellodb> select * from students s where exists (select * from teachers t w
ere s.`Tea -> cherID`=t.`TID`);
+-------+------------+-----+--------+---------+-----------+
| StuID | Name       | Age | Gender | ClassID | TeacherID |
+-------+------------+-----+--------+---------+-----------+
| 1     | Shi Zhongyu | 22  | M      | 2       | 3         |
| 4     | Ding Dian   | 32  | M      | 4       | 4         |
| 5     | Yu Yutong   | 26  | M      | 3       | 1         |
+-------+------------+-----+--------+---------+-----------+
#说明：

1. EXISTS (或 NOT EXISTS) 用在 where之后，且后面紧跟子查询语句（带括号）
2. EXISTS (或 NOT EXISTS) 只关心子查询有没有结果,并不关心子查询的结果具体是什么
3. 上述语句把students的记录逐条代入到Exists后面的子查询中，如果子查询结果集不为空，即说明在，那么这条students的记录出现在最终结果集，否则被排除

#查询学生表中无对应老师的学生
mysql root@(none):hellodb> select * from students s where not exists(select * from teachers t
where s.` -> TeacherID`=t.tid);
+-------+--------------+-----+--------+---------+-----------+
| StuID | Name         | Age | Gender | ClassID | TeacherID |
+-------+--------------+-----+--------+---------+-----------+
| 2     | Shi Potian   | 22  | M      | 1       | 7         |
| 3     | Xie Yanke    | 53  | M      | 2       | 16        |
| 6     | Shi Qing     | 46  | M      | 5       |     |
| 7     | Xi Ren       | 19  | F      | 3       |     |
| 8     | Lin Daiyu    | 17  | F      | 7       |     |
| 9     | Ren Yingying | 20  | F      | 6       |     |
| 10    | Yue Lingshan | 19  | F      | 3       |     |
| 11    | Yuan Chengzhi | 23  | M      | 6       |     |
| 12    | Wen Qingqing | 19  | F      | 1       |     |
| 13    | Tian Boguang | 33  | M      | 2       |     |
| 14    | Lu Wushuang  | 17  | F      | 3       |     |
```

```
| 15   | Duan Yu       | 19  | M      | 4       |      |
| 16   | Xu Zhu        | 21  | M      | 1       |      |
| 17   | Lin Chong     | 25  | M      | 4       |      |
| 18   | Hua Rong      | 23  | M      | 7       |      |
| 19   | Xue Baochai   | 18  | F      | 6       |      |
| 20   | Diao Chan     | 19  | F      | 7       |      |
| 21   | Huang Yueying | 22  | F      | 6       |      |
| 22   | Xiao Qiao     | 20  | F      | 1       |      |
| 23   | Ma Chao       | 23  | M      | 4       |      |
| 24   | Xu Xian       | 27  | M      |         |      |
| 25   | Sun Dasheng   | 100 | M      |         |      |
+------+---------------+-----+--------+---------+-----------+
```

4. 用于FROM子句中的子查询

使用格式：

```
SELECT tb_alias.col1,... FROM (SELECT clause) AS tb_alias WHERE Clause;
```

范例：

```
mysql root@(none):hellodb> select s.classid,s.aage from (select classid,avg(age) as aage from
tudents
-> where `ClassID` is not null group by classid) as s where s.aage>30;
```

范例：子查询

```
#子查询：select 的执行结果，被其它SQL调用
#查询学生表中学生年龄大于老师平均年龄的学生
mysql root@(none):hellodb> select `StuID`,`Name`,`Age` from students where age > (select a
g(age) from
-> teachers);
```

范例：子查询用于更新表

```
mysql root@(none):hellodb> update teachers set age=(select avg(age) from students) where
TID`=4;
mysql root@(none):hellodb> select * from teachers;
+-----+---------------+-----+--------+
| TID | Name          | Age | Gender |
+-----+---------------+-----+--------+
| 1   | Song Jiang    | 45  | M      |
| 2   | Zhang Sanfeng | 94  | M      |
| 3   | Miejue Shitai | 77  | F      |
| 4   | Lin Chaoying  | 27  | F      |
+-----+---------------+-----+--------+
```

3.7.2.2 联合查询

联合查询 Union 实现的条件,多个表的字段数量相同,字段名和数据类型可以不同,但一般数据类型是相的

```
SELECT Name,Age FROM students UNION SELECT Name,Age FROM teachers;
```

范例：联合查询

```
#多表联合查询，联合查询的表必须字段数相同
select tid as id,name,age,gender from teachers union select stuid,name,age,gender from stud
nts;
#如果是相同的表不会显示相同的信息
select * from teachers union select * from teachers;
#如果想显示相同的信息需要在union后面加all
select * from teachers union all select * from teachers;
```

范例：去重

```
#除去查询结果中的重复记录的distinct
select distinct * from teachers;
#联合查询默认是去重的union all不去重
```

3.7.2.3 交叉连接

cross join 即多表的记录之间做笛卡尔乘积组合，并且多个表的列横向合并相加, "雨露均沾"

比如: 第一个表3行4列,第二个表5行6列,cross join后的结果为3*5=15行,4+6=10列

交叉连接生成的记录可能会非常多,建议慎用

范例：交叉连接

```
#横向合并，交叉连接（横向笛卡尔）
#俩种写法
select * from students cross join teachers;
select * from students,teachers;
```

3.7.2.4 内连接

inner join 内连接取多个表的交集

范例：内连接

```
#内连接inner join
#查询学生表中代课老师id和老师表老师id相同的数据把他们连接起来，俩种写法
select * from students inner join teachers on students.teacherid=teachers.tid;
select * from students,teachers where students.teacherid=teachers.tid;
#如果表定义了别名，原表名将无法使用
select stuid,s.name as student_name,tid,t.name as teacher_name from students as s inner join
eachers as t on s.teacherid=t.tid;
#查询学生性别和老师性别不相等的，由于老师有多个学生有多个每个学生都会与老师进行比较不相
的都会显示
select s.name 学生姓名,s.age 学生年龄,s.gender 学生性别,t.name 老师姓名,t.age 老师年龄,t.gende
 老师性别 from students s,teachers t where s.`Gender` <> t.`Gender`;
#内连接后过滤数据
#查询学生表有代课老师的学生并且年龄大于30的人与想对应的老师数据进行内连接
select * from students s inner join teachers t on s.`TeacherID`=t.tid and s.age >30;
```

自然连接

- 当源表和目标表共享相同名称的列时，就可以在它们之间执行自然连接，而无需指定连接列。
- 在使用纯自然连接时，如没有相同的列时，会产生交叉连接（笛卡尔乘积）
- 语法：(SQL:1999)SELECT table1.column, table2.column FROM table1 NATURAL JOIN table2;

3.7.2.5 左和右外连接

范例：左，右外连接

```
#左外连接
select * from students as s left outer join teachers as t on s.`TeacherID`=t.tid;
#左外连接扩展
select * from students as s left outer join teachers as t on s.`TeacherID`=t.tid where t.tid is null;
#多个条件的左外连接
select * from students as s left outer join teachers as t on s.`TeacherID`=t.tid and s.`TeacherID`
s null;
#先左外连接，再过滤
select * from students as s left outer join teachers as t on s.`TeacherID`=t.tid where s.`Teacher
D` is null;
#右外连接
select * from students s right outer join teachers t on s.`TeacherID`=t.tid;
#右外连接的扩展用法
select * from students s right outer join teachers t on s.`TeacherID`=t.tid where s.`TeacherID` is
null;
```

3.7.2.6 完全外连接

MySQL 不支持完全外连接full outer join语法

```
#MySQL不支持完全外连接 full outer join,利用以下方式法代替
select * from students left join teachers on students.`TeacherID`=teachers.tid union select * fr
m students right join teachers on students.`TeacherID`=teachers.tid;
```

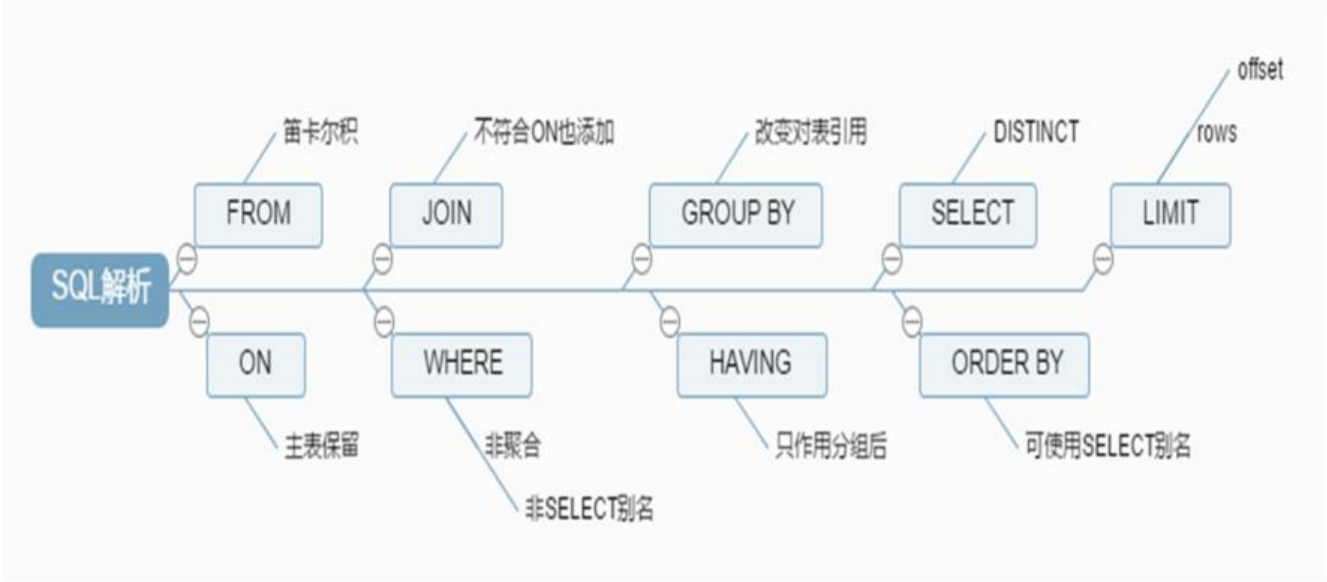
3.7.2.7 自连接

自连接, 即表自身连接自身

范例：自连接

```
#自连接
select * from teachers;
select e.name,l.name from teachers as e inner join teachers as l on e.`TID`=l.tid;
```

3.7.3 SELECT 语句处理的顺序



查询执行路径中的组件：查询缓存、解析器、预处理器、优化器、查询执行引擎、存储引擎

SELECT语句的执行流程：

FROM Clause --> WHERE Clause --> GROUP BY --> HAVING Clause -->SELECT -->ORDER BY --> LIMIT