

05 | 检索数据：你还在SELECT * 么？

2019-06-21 陈旻



今天我们主要学习如何使用**SQL**检索数据。如果你已经有了一定的**SQL**基础，这节课可以跳过，也可以把它当做是个快速的复习。

SELECT可以说是**SQL**中最常用的语句了。你可以把**SQL**语句看作是英语语句，**SELECT**就是**SQL**中的关键字之一，除了**SELECT**之外，还有**INSERT**、**DELETE**、**UPDATE**等关键字，这些关键字是**SQL**的保留字，这样可以很方便地帮助我们分析理解**SQL**语句。我们在定义数据库表名、字段名和变量名时，要尽量避免使用这些保留字。

SELECT的作用是从一个表或多个表中检索出想要的数据库行。今天我主要讲解**SELECT**的基础查询，后面我会讲解如何通过多个表的连接操作进行复杂的查询。

在这篇文章中，你需要重点掌握以下几方面的内容：

1. **SELECT**查询的基础语法；
2. 如何排序检索数据；
3. 什么情况下用**SELECT***，如何提升**SELECT**查询效率？

SELECT查询的基础语法

SELECT可以帮助我们从一个表或多个表中进行数据查询。我们知道一个数据表是由列（字段名）和行（数据行）组成的，我们要返回满足条件的数据库行，就需要在**SELECT**后面加上我们想要查询的列名，可以是一列，也可以是多个列。如果你不知道所有列名都有什么，也可以检索所

有列。

我创建了一个王者荣耀英雄数据表，这张表里一共有69个英雄，23个属性值（不包括英雄名name）。SQL文件见[Github地址](#)。

id	name	hp_max	hp_growth	hp_start	role_assist	birthdate
10000	夏侯惇	7350	288.8	3307		战士	2016-07-19
10001	钟无艳	7000	275	3150		坦克	
10002	张飞	8341	329	3450		辅助	
.....
10068	百里守约	5611	185	3019		刺客	2017-08-08

数据表中这24个字段（除了id以外），分别代表的含义见下图。

name 英雄名称	hp_max 最大生命	hp_growth 生命成长	hp_start 初始生命
mp_max 最大法力	mp_growth 法力成长	mp_start 初始法力	attack_max 最高物攻
attack_growth 物攻成长	attack_start 初始物攻	defense_max 最大物防	defense_growth 物防成长
defense_start 初始物防	hp_5s_max 最大每5秒回血	hp_5s_growth 每5秒回血成长	hp_5s_start 初始每5秒回血
mp_5s_max 最大每5秒回蓝	mp_5s_growth 每5秒回蓝成长	mp_5s_start 初始每5秒回蓝	attack_range 攻击范围
attack_speed_max 最大攻速	role_main 主要定位	role_assist 次要定位	birthdate 上线时间

查询列

如果我们想要对数据表中的某一系列进行检索，在SELECT后面加上这个列的字段名即可。比如我们想要检索数据表中都有哪些英雄。

```
SQL: SELECT name FROM heros
```

运行结果（69条记录）见下图，你可以看到这样就等于单独输出了name这一列。

name
夏侯惇
钟无艳
.....
百里守约

我们也可以对多个列进行检索，在列名之间用逗号(,)分割即可。比如我们想要检索有哪些英雄，他们的最大生命、最大法力、最大物攻和最大物防分别是多少。

```
SQL: SELECT name, hp_max, mp_max, attack_max, defense_max FROM heros
```

运行结果（69条记录）：

name	hp_max	mp_max	attack_max	defense_max
夏侯惇	7350	1746	321	397
钟无艳	7000	1760	318	409
张飞	8341	100	301	504
.....
百里守约	5611	1784	410	329

这个表中一共有25个字段，除了id和英雄名name以外，还存在23个属性值，如果我们记不住所有的字段名称，可以使用SELECT * 帮我们检索出所有的列：

```
SQL: SELECT * FROM heros
```

运行结果（69条记录）：

id	name	hp_max	hp_growth	hp_start	role_assist	birthdate
10000	夏侯惇	7350	288.8	3307		战士	2016-07-19
10001	钟无艳	7000	275	3150		坦克	
10002	张飞	8341	329	3450		辅助	
.....
10068	百里守约	5611	185	3019		刺客	2017-08-08

我们在做数据探索的时候，SELECT * 还是很有用的，这样我们就不需要写很长的SELECT语句了。但是在生产环境时要尽量避免使用SELECT *，具体原因我会在后面讲。

起别名

我们在使用**SELECT**查询的时候，还有一些技巧可以使用，比如你可以给列名起别名。我们在进行检索的时候，可以给英雄名、最大生命、最大法力、最大物攻和最大物防等取别名：

```
SQL: SELECT name AS n, hp_max AS hm, mp_max AS mm, attack_max AS am, defense_max AS dm FROM t
```

运行结果和上面多列检索的运行结果是一样的，只是将列名改成了**n**、**hm**、**mm**、**am**和**dm**。当然这里的列别名只是举例，一般来说起别名的作用是对原有名称进行简化，从而让**SQL**语句看起来更精简。同样我们也可以对表名称起别名，这个在多表连接查询的时候会用到。

查询常数

SELECT查询还可以对常数进行查询。对的，就是在**SELECT**查询结果中增加一列固定的常数列。这列的取值是我们指定的，而不是从数据表中动态取出的。你可能会问为什么我们还要对常数进行查询呢？**SQL**中的**SELECT**语法的确提供了这个功能，一般来说我们只从一个表中查询数据，通常不需要增加一个固定的常数列，但如果我们想整合不同的数据源，用常数列作为这个表的标记，就需要查询常数。

比如说，我们想对**heros**数据表中的英雄名进行查询，同时增加一列字段**platform**，这个字段固定值为“王者荣耀”，可以这样写：

```
SQL: SELECT '王者荣耀' as platform, name FROM heros
```

运行结果：（69条记录）

platform	name
王者荣耀	夏侯惇
王者荣耀	钟无艳
王者荣耀	张飞
王者荣耀
王者荣耀	百里守约

在这个**SQL**语句中，我们虚构了一个**platform**字段，并且把它设置为固定值“王者荣耀”。

需要说明的是，如果常数是个字符串，那么使用单引号（**'**）就非常重要了，比如**'王者荣耀'**。单引号说明引号中的字符串是个常数，否则**SQL**会把王者荣耀当成列名进行查询，但实际上数据表里没有这个列名，就会引起错误。如果常数是英文字母，比如**'WZRY'**也需要加引号。如果常数是数字，就可以直接写数字，不需要单引号，比如：

```
SQL: SELECT 123 as platform, name FROM heros
```

运行结果：（69条记录）

platform	name
123	夏侯惇
123	钟无艳
123	张飞
123
123	百里守约

去除重复行

关于单个表的**SELECT**查询，还有一个非常实用的操作，就是从结果中去掉重复的行。使用的关键字是**DISTINCT**。比如我们想要看下**heros**表中关于攻击范围的取值都有哪些：

```
SQL: SELECT DISTINCT attack_range FROM heros
```

这是运行结果（2条记录），这样我们就能直观地看到攻击范围其实只有两个值，那就是近战和远程。

attack_range
近战
远程

如果我们带上英雄名称，会是怎样呢：

```
SQL: SELECT DISTINCT attack_range, name FROM heros
```

运行结果（69条记录）：

attack_range	name
近战	夏侯惇
近战	钟无艳
近战	张飞
.....
远程	百里守约

这里有两点需要注意：

1. **DISTINCT**需要放到所有列名的前面，如果写成**SELECT name, DISTINCT attack_range FROM heros**会报错。
2. **DISTINCT**其实是对后面所有列名的组合进行去重，你能看到最后的结果是**69**条，因为这**69**个英雄名称不同，都有攻击范围（**attack_range**）这个属性值。如果你想要看都有哪些不同的攻击范围（**attack_range**），只需要写**DISTINCT attack_range**即可，后面不需要再加其他的列名了。

如何排序检索数据

当我们检索数据的时候，有时候需要按照某种顺序进行结果的返回，比如我们想要查询所有的英雄，按照最大生命从高到底的顺序进行排列，就需要使用**ORDER BY**子句。使用**ORDER BY**子句有以下几个点需要掌握：

1. 排序的列名：**ORDER BY**后面可以有一个或多个列名，如果是多个列名进行排序，会按照后面第一个列先进行排序，当第一列的值相同的时候，再按照第二列进行排序，以此类推。
2. 排序的顺序：**ORDER BY**后面可以注明排序规则，**ASC**代表递增排序，**DESC**代表递减排序。如果没有注明排序规则，默认情况下是按照**ASC**递增排序。我们很容易理解**ORDER BY**对数值类型字段的排序规则，但如果排序字段类型为文本数据，就需要参考数据库的设置方式了，这样才能判断**A**是在**B**之前，还是在**B**之后。比如使用**MySQL**在创建字段的时候设置为**BINARY**属性，就代表区分大小写。
3. 非选择列排序：**ORDER BY**可以使用非选择列进行排序，所以即使在**SELECT**后面没有这个列名，你同样可以放到**ORDER BY**后面进行排序。
4. **ORDER BY**的位置：**ORDER BY**通常位于**SELECT**语句的最后一条子句，否则会报错。

在了解了**ORDER BY**的使用语法之后，我们来看下如何对**heros**数据表进行排序。

假设我们想要显示英雄名称及最大生命值，按照最大生命值从高到低的方式进行排序：

```
SQL: SELECT name, hp_max FROM heros ORDER BY hp_max DESC
```

运行结果（69条记录）：

name	hp_max
廉颇	9328
白起	8638
程咬金	8611
.....
武则天	5037

如果想要显示英雄名称及最大生命值，按照第一排序最大法力从低到高，当最大法力值相等的时候则按照第二排序进行，即最大生命值从高到低的方式进行排序：

```
SQL: SELECT name, hp_max FROM heros ORDER BY mp_max, hp_max DESC
```

运行结果：（69条记录）

name	hp_max
程咬金	8611
亚瑟	8050
曹操	7473
.....
妲己	5824

约束返回结果的数量

另外在查询过程中，我们可以约束返回结果的数量，使用**LIMIT**关键字。比如我们想返回英雄名称及最大生命值，按照最大生命值从高到低排序，返回**5**条记录即可。

```
SQL: SELECT name, hp_max FROM heros ORDER BY hp_max DESC LIMIT 5
```

运行结果（5条记录）：

name	hp_max
廉颇	9328
白起	8638
程咬金	8611
刘禅	8581
牛魔	8476

有一点需要注意，约束返回结果的数量，在不同的DBMS中使用的关键字可能不同。在MySQL、PostgreSQL、MariaDB和SQLite中使用LIMIT关键字，而且需要放到SELECT语句的最后面。如果是SQL Server和Access，需要使用TOP关键字，比如：

```
SQL: SELECT TOP 5 name, hp_max FROM heros ORDER BY hp_max DESC
```

如果是DB2，使用FETCH FIRST 5 ROWS ONLY这样的关键字：

```
SQL: SELECT name, hp_max FROM heros ORDER BY hp_max DESC FETCH FIRST 5 ROWS ONLY
```

如果是Oracle，你需要基于ROWNUM来统计行数：

```
SQL: SELECT name, hp_max FROM heros WHERE ROWNUM <=5 ORDER BY hp_max DESC
```

需要说明的是，这条语句是先取出来前5条数据行，然后再按照hp_max从高到低的顺序进行排序。但这样产生的结果和上述方法的并不一样。我会在后面讲到子查询，你可以使用SELECT name, hp_max FROM (SELECT name, hp_max FROM heros ORDER BY hp_max) WHERE ROWNUM <=5得到与上述方法一致的结果。

约束返回结果的数量可以减少数据表的网络传输量，也可以提升查询效率。如果我们知道返回结果只有1条，就可以使用LIMIT 1，告诉SELECT语句只需要返回一条记录即可。这样的好处就是SELECT不需要扫描完整的表，只需要检索到一条符合条件的记录即可返回。

SELECT的执行顺序

查询是RDBMS中最频繁的操作。我们在理解SELECT语法的时候，还需要了解SELECT执行时的底层原理。只有这样，才能让我们对SQL有更深刻的认识。

其中你需要记住SELECT查询时的两个顺序：

1.关键字的顺序是不能颠倒的：


```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ...
```

2.SELECT语句的执行顺序（在MySQL和Oracle中，SELECT执行顺序基本相同）：

FROM > WHERE > GROUP BY > HAVING > SELECT的字段 > DISTINCT > ORDER BY > LIMIT

比如你写了一个SQL语句，那么它的关键字顺序和执行顺序是下面这样的：

```
SELECT DISTINCT player_id, player_name, count(*) as num #顺序5
FROM player JOIN team ON player.team_id = team.team_id #顺序1
WHERE height > 1.80 #顺序2
GROUP BY player.team_id #顺序3
HAVING num > 2 #顺序4
ORDER BY num DESC #顺序6
LIMIT 2 #顺序7
```

在SELECT语句执行这些步骤的时候，每个步骤都会产生一个虚拟表，然后将这个虚拟表传入下一个步骤中作为输入。需要注意的是，这些步骤隐含在SQL的执行过程中，对于我们来说是不可见的。

我来详细解释一下SQL的执行原理。

首先，你可以注意到，SELECT是先执行FROM这一步的。在这个阶段，如果是多张表联查，还会经历下面的几个步骤：

1. 首先先通过CROSS JOIN求笛卡尔积，相当于得到虚拟表 vt（virtual table）1-1；
2. 通过ON进行筛选，在虚拟表vt1-1的基础上进行筛选，得到虚拟表 vt1-2；
3. 添加外部行。如果我们使用的是左连接、右链接或者全连接，就会涉及到外部行，也就是在虚拟表vt1-2的基础上增加外部行，得到虚拟表vt1-3。

当然如果我们操作的是两张以上的表，还会重复上面的步骤，直到所有表都被处理完为止。这个过程得到的是我们的原始数据。

当我们拿到了查询数据表的原始数据，也就是最终的虚拟表vt1，就可以在此基础上再进行WHERE阶段。在这个阶段中，会根据vt1表的结果进行筛选过滤，得到虚拟表vt2。

然后进入第三步和第四步，也就是GROUP和 HAVING阶段。在这个阶段中，实际上是在虚拟表

vt2的基础上进行分组和分组过滤，得到中间的虚拟表vt3和vt4。

当我们完成了条件筛选部分之后，就可以筛选表中提取的字段，也就是进入到**SELECT**和**DISTINCT**阶段。

首先在**SELECT**阶段会提取想要的字段，然后在**DISTINCT**阶段过滤掉重复的行，分别得到中间的虚拟表vt5-1和vt5-2。

当我们提取了想要的字段数据之后，就可以按照指定的字段进行排序，也就是**ORDER BY**阶段，得到虚拟表vt6。

最后在vt6的基础上，取出指定行的记录，也就是**LIMIT**阶段，得到最终的结果，对应的是虚拟表vt7。

当然我们在写**SELECT**语句的时候，不一定存在所有的关键字，相应的阶段就会省略。

同时因为**SQL**是一门类似英语的结构化查询语言，所以我们在写**SELECT**语句的时候，还要注意相应的关键字顺序，所谓底层运行的原理，就是我们刚才讲到的执行顺序。

什么情况下用**SELECT***，如何提升**SELECT**查询效率？

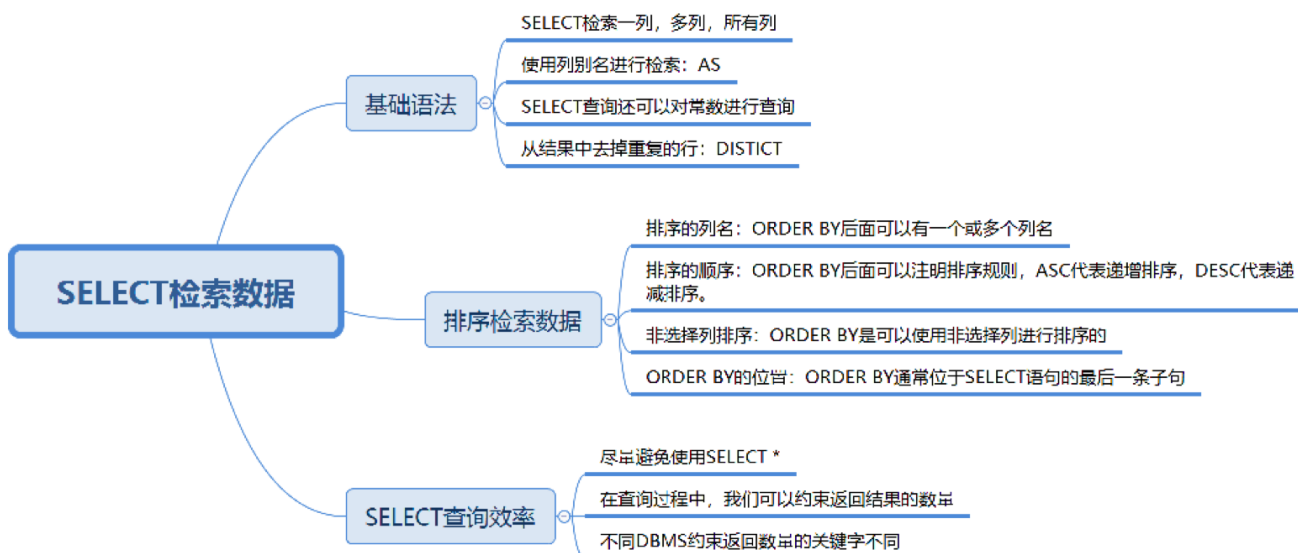
当我们初学**SELECT**语法的时候，经常会使用**SELECT***，因为使用方便。实际上这样也增加了数据库的负担。所以如果我们不需要把所有列都检索出来，还是先指定出所需的列名，因为写清列名，可以减少数据表查询的网络传输量，而且考虑到在实际的工作中，我们往往不需要全部的列名，因此你需要养成良好的习惯，写出所需的列名。

如果我们只是练习，或者对数据表进行探索，那么是可以使用**SELECT***的。它的查询效率和把所有列名都写出来再进行查询的效率相差并不大。这样可以方便你对数据表有个整体的认知。但是在生产环境下，不推荐你直接使用**SELECT***进行查询。

总结

今天我对**SELECT**的基础语法进行了讲解，**SELECT**是**SQL**的基础。但不同阶段看**SELECT**都会有新的体会。当你第一次学习的时候，关注的往往是如何使用它，或者语法是否正确。再看的时候，可能就会更关注**SELECT**的查询效率，以及不同**DBMS**之间的差别。

在我们的日常工作中，很多人都可以写出**SELECT**语句，但是执行的效率却相差很大。产生这种情况的原因主要有两个，一个是习惯的培养，比如大部分初学者会经常使用**SELECT***，而好的习惯则是只查询所需要的列；另一个对**SQL**查询的执行顺序及查询效率的关注，比如当你知道只有1条记录的时候，就可以使用**LIMIT 1**来进行约束，从而提升查询效率。



最后留两道思考题吧，我今天对单表的**SELECT**查询进行了讲解，你之前可能也有学习使用的经验，可以说下你对**SELECT**使用的理解吗？另外，我今天使用**heros**数据表进行了举例，请你编写**SQL**语句，对英雄名称和最大法力进行查询，按照最大生命从高到低排序，只返回**5**条记录即可。你可以说明下使用的**DBMS**及相应的**SQL**语句。

欢迎你把这篇文章分享给你的朋友或者同事，与他们一起来分析一下王者荣耀的数据，互相切磋交流。

 极客时间

SQL 必知必会

从入门到数据实战

陈 旻

清华大学计算机博士



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。



SELECT COUNT(*) > SELECT COUNT(1) > SELECT COUNT(具体字段)

之前看到的，好像Mysql对count(*)做了单独的优化

2019-06-22

作者回复

关于COUNT()的效率是一个很好的问题，欢迎探讨：

在MySQL InnoDB存储引擎中，COUNT(*)和COUNT(1)都是对所有结果进行的COUNT。如果有WHERE子句，则是对所有符合筛选条件的数据行进行统计。如果没有WHERE子句，则是对数据表的数据行数进行统计。

因此COUNT(*)和COUNT(1)本质上没有区别，执行的复杂度都是O(N)，也就是采用全表扫描，进行循环+计数的方式进行统计。

如果是MySQL MyISAM存储引擎，统计数据表的行数只需要O(1)复杂度，这是因为每张MyISAM的数据表都有一个meta信息有存储了row_count值。而一致性由表级锁来保证。而InnoDB支持事务，采用行级锁和MVCC机制，所以无法像MyISAM一样，只维护一个row_count变量。因此就需要采用扫描全表，进行循环+计数的方式来完成统计。

需要注意的是，在实际执行中COUNT(*)和COUNT(1)执行时间可能略有差别，不过你还是可以把这两个在执行效率上看成是相等的。

另外在InnoDB引擎中，如果是采用COUNT(*)和COUNT(1)来统计数据行数，要尽量采用二级索引。

因为主键采用的索引是聚簇索引，聚簇索引包含的信息多，明显会大于二级索引（非聚簇索引）。

对于查找具体的行来说，采用主键索引效率更高。而对于COUNT(*)和COUNT(1)这种，不需要查找具体的行，只是统计行数来说，系统会自动采用占用空间更小的二级索引来进行统计。

如果有多个二级索引的时候，会使用key_len小的二级索引进行扫描。当没有二级索引的时候，才会采用主键索引来进行统计。

优化总结：

1、一般情况下：COUNT(*) = COUNT(1) > COUNT(字段)

所以尽量使用COUNT(*)，当然如果你要统计的是就是某个字段的非空数据行数，那另当别论。毕竟执行效率比较的前提是要结果一样才行。

2、如果要统计COUNT(*)，尽量在数据表上建立二级索引，系统会自动采用key_len小的二级索引进行扫描，这样当我们使用SELECT COUNT(*)的时候效率就会提升，有时候提升几倍甚至更高都是有可能的。

2019-06-23



做一个搬运工，下面是从老师GitHub上找到的建表语句：

```
SET NAMES utf8mb4;
```

```
SET FOREIGN_KEY_CHECKS = 0;
```

```
-- -----  
-- Table structure for heros  
-- -----
```

```
DROP TABLE IF EXISTS `heros`;  
CREATE TABLE `heros` (  
  `name` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,  
  `hp_max` float NULL DEFAULT NULL,  
  `hp_growth` float NULL DEFAULT NULL,  
  `hp_start` float NULL DEFAULT NULL,  
  `mp_max` float NULL DEFAULT NULL,  
  `mp_growth` float NULL DEFAULT NULL,  
  `mp_start` float NULL DEFAULT NULL,  
  `attack_max` float NULL DEFAULT NULL,  
  `attack_growth` float NULL DEFAULT NULL,  
  `attack_start` float NULL DEFAULT NULL,  
  `defense_max` float NULL DEFAULT NULL,  
  `defense_growth` float NULL DEFAULT NULL,  
  `defense_start` float NULL DEFAULT NULL,  
  `hp_5s_max` float NULL DEFAULT NULL,  
  `hp_5s_growth` float NULL DEFAULT NULL,  
  `hp_5s_start` float NULL DEFAULT NULL,  
  `mp_5s_max` float NULL DEFAULT NULL,  
  `mp_5s_growth` float NULL DEFAULT NULL,  
  `mp_5s_start` float NULL DEFAULT NULL,  
  `attack_speed_max` float NULL DEFAULT NULL,  
  `attack_range` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,  
  `role_main` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,  
  `role_assist` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,  
  `birthdate` datetime(0) NULL DEFAULT NULL,  
  PRIMARY KEY (`name`) USING BTREE  
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

2019-06-22



ack

👍 13

老师好，请问能把建表的sql给出来吗？

2019-06-21

select id from table where id=5

执行这条语句时，mysql会判断是否有缓存，有的话直接返回，如果没有则调用引擎接口查询数据，引擎比如说innodb会判断这条数据是否在内存中存在，如果存在直接返回给mysql的server层，如果不存在则会到磁盘里把id=5的这条数据所在的整个数据页都读取到内存中，这个内存就是innodb buffer pool，是一块固定大小的地方，可以由参数innodb_buffer_pool_size设置大小，既然是一块固定大小的地方，那么就要有淘汰机制，mysql是在lru算法的基础上做了优化，简单的来说就是最久未被使用的数据会被淘汰掉

2019-06-21



hogen

5

-- 注释版本

```
CREATE TABLE `heros` (
  `id` SMALLINT NOT NULL PRIMARY KEY AUTO_INCREMENT COMMENT '主键',
  `name` VARCHAR(255) DEFAULT NULL COMMENT '英雄名称',
  `hp_max` FLOAT NULL DEFAULT NULL COMMENT '最大生命',
  `hp_growth` FLOAT NULL DEFAULT NULL COMMENT '生命成长',
  `hp_start` FLOAT NULL DEFAULT NULL COMMENT '初始生命',
  `mp_max` FLOAT NULL DEFAULT NULL COMMENT '最大法力',
  `mp_growth` FLOAT NULL DEFAULT NULL COMMENT '法力成长',
  `mp_start` FLOAT NULL DEFAULT NULL COMMENT '初始法力',
  `attack_max` FLOAT NULL DEFAULT NULL COMMENT '最高物攻',
  `attack_growth` FLOAT NULL DEFAULT NULL COMMENT '物攻成长',
  `attack_start` FLOAT NULL DEFAULT NULL COMMENT '初始物攻',
  `defense_max` FLOAT NULL DEFAULT NULL COMMENT '最大物防',
  `defense_growth` FLOAT NULL DEFAULT NULL COMMENT '物防成长',
  `defense_start` FLOAT NULL DEFAULT NULL COMMENT '初始物防',
  `hp_5s_max` FLOAT NULL DEFAULT NULL COMMENT '最大每5秒回血',
  `hp_5s_growth` FLOAT NULL DEFAULT NULL COMMENT '每5秒回血成长',
  `hp_5s_start` FLOAT NULL DEFAULT NULL COMMENT '初始每5秒回血',
  `mp_5s_max` FLOAT NULL DEFAULT NULL COMMENT '最大每5秒回蓝',
  `mp_5s_growth` FLOAT NULL DEFAULT NULL COMMENT '每5秒回蓝成长',
  `mp_5s_start` FLOAT NULL DEFAULT NULL COMMENT '初始每5秒回蓝',
  `attack_speed_max` FLOAT NULL DEFAULT NULL COMMENT '最大攻速',
  `attack_range` VARCHAR(255) COMMENT '攻击范围',
  `role_main` VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL COMMENT '主要定位',
  `role_assist` VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL COMMENT '次要定位',
  `birthdate` DATETIME(0) NULL DEFAULT NULL COMMENT '上线时间'
) ENGINE = INNODB DEFAULT CHARSET=UTF8 COMMENT='王者荣耀数据表';
```

2019-06-22



Shame

3

先交作业 `select name,mp_max from heros order by hp_max desc limit 5;`
然后就是楼下一个同学问的问题，我也有些疑惑，就是这个
`SELECT DISTINCT player_id, player_name, count(*) as num # 顺序 5`
`FROM player JOIN team ON player.team_id = team.team_id # 顺序 1`
`WHERE height > 1.80 # 顺序 2`
`GROUP BY player.team_id # 顺序 3`
`HAVING num > 2 # 顺序 4`
`ORDER BY num DESC # 顺序 6`
`LIMIT 2 # 顺序 7`

对于这个语句，我还有一点疑问：既然HAVING的执行是在SELECT之前的，那么按理说在执行HAVING的时候SELECT中的count(*)应该还没有被计算出来才对啊，为什么在HAVING中就直接使用了num>2这个条件呢？

希望老师百忙之中能抽空帮忙解释一下，谢谢老师

2019-06-27

作者回复

很好的问题，实际上在Step4和Step5之间，还有个聚集函数的计算。

如果加上这个计算过程，完整的顺序是：

- 1、FROM子句组装数据
- 2、WHERE子句进行条件筛选
- 3、GROUP BY分组
- 4、使用聚集函数进行计算；
- 5、HAVING筛选分组；
- 6、计算所有的表达式；
- 7、SELECT 的字段；
- 8、ORDER BY排序
- 9、LIMIT筛选

所以中间有两个过程是需要计算的：聚集函数 和 表达式。其余是关键字的执行顺序，如文章所示。

2019-06-28



Samson

👍 3

`SELECT DISTINCT player_id, player_name, count(*) as num # 顺序 5`
`FROM player JOIN team ON player.team_id = team.team_id # 顺序 1`
`WHERE height > 1.80 # 顺序 2`
`GROUP BY player.team_id # 顺序 3`
`HAVING num > 2 # 顺序 4`
`ORDER BY num DESC # 顺序 6`
`LIMIT 2 # 顺序 7`

对于这个语句，我还有一点疑问：既然HAVING的执行是在SELECT之前的，那么按理说在执行HAVING的时候SELECT中的count(*)应该还没有被计算出来才对啊，为什么在HAVING中就直

接使用了num>2这个条件呢？

2019-06-24



石维康

👍 3

作业: `SELECT name, mp_max FROM heros ORDER BY hp_max DESC LIMIT 5;`
MySQL数据库

2019-06-21



极客时间

👍 2

老师 我有一个疑问 场景是这样的，我有三张表，一张表存储文章posts，一张表存储文章标签tags，

一篇文章可以有多个标签，一个标签可以被多个文章拥有，文章和标签是多对多的关系，

此时我又增加了一个关系表post_tag,这个表只有两个字段，post_id和tag_id.

我现在有一个需求 查询出所有文章，查询出的文章数据中每篇文章都有一个tags属性，这个属性包含所有这篇文章的标签信息，这个查询应该怎么做呢？或者说sql只能做一部分，然后在通过其他脚本语言再处理呢？

我将问题发在了 segmentfault 链接地址 <https://segmentfault.com/q/1010000019472412>

2019-06-26



Samson

👍 2

老师，可以说下SELECT语句执行原理那个视例中HAVING关键字的作用嘛？

2019-06-24

作者回复

HAVING一般配合GROUP BY使用，作为筛选分组的条件。作用实际上和WHERE一样，都适用于限定条件。只是WHERE子句用于对查询结果的分组前，通过WHERE来过滤。而HAVING子句用于筛选满足条件的组，用于在分组之后进行过滤。这个我在后面也会讲到。

2019-06-24



hlz-123

👍 2

数据库，MySQL8.0

`SELECT name as '姓名',mp_max as '最大法力' FROM heros ORDER BY hp_max LIMIT 5;`

2019-06-23



crazypokerk

👍 2

SELECT是所有查询关键字最后一个执行的关键字。

2019-06-21



mickey

👍 2

1.SELECT就是告诉数据库要选什么字段

2.

1)MySQL、PostgreSQL、MariaDB和SQLite:

```
SELECT name, mp_max FROM heros ORDER BY hp_max DESC limit 5;
```

2)SQL Server和Access:

```
SELECT TOP 5 name, mp_max FROM heros ORDER BY hp_max DESC
```

3)DB2:

```
SELECT name, mp_max FROM heros ORDER BY hp_max DESC FETCH FIRST 5 ROWS ONLY;
```

4)Oracle:

```
SELECT name, mp_max FROM (SELECT name, mp_max FROM heros ORDER BY hp_max) as t WHERE ROWNUM <=5
```

2019-06-21



月牙天冲

👍 2

1.作业: `SELECT name,mp_max FROM heros ORDER BY hp_max DECS LIMIT 5;`

2.老师能提供下创建表的sql语句么, 谢谢。

2019-06-21



业余草

👍 2

`create table` 还没学吧, 我是小白, 教一下 `create table` 或者 `create table like`。就单拿 `select` 说, 这章内容也不全啊, `group`, `having` 等都漏掉了

2019-06-21



菜鸡小王子

👍 2

希望老师在讲`select`的基础语法时 能稍微带一点底层原理啦 比如`select`的执行顺序 这样理解是不是可以更加透彻一点

2019-06-21

编辑回复

相关内容已经更新到了文章里, 感谢您的反馈。

2019-06-21



cricket1981

👍 1

```
mysql> SELECT name, mp_max FROM heros ORDER BY hp_max DESC LIMIT 5;
```

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

```
| name | mp_max |
```

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

```
| 廉颇 | 1708 |
```

```
| 白起 | 1666 |
```

```
| 程咬金 | 0 |
```

```
| 刘禅 | 1694 |
```

```
| 牛魔 | 1926 |
```

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

```
5 rows in set (0.00 sec)
```

2019-06-26



UJ00N00

👍 1

SELECT name, mp_max FROM heros ORDER BY hp_max ASC LIMIT 5;

2019-06-25

作者回复

应该是按照最大生命值从高到低进行排序，使用DESC

2019-06-25



©™

👍 1

老师，现在有a,b,c三张表，我先用a inner join b，接着再用a left join c，这样做可以达到获取a,b关联数据的目的吗？

2019-06-24



时间是最真的答案

👍 1

MySQL

SELECT `name`,mp_max FROM heros ORDER BY hp_max DESC LIMIT 5

2019-06-24

作者回复

正确，同时也考虑到了给查询字段使用反引号

2019-06-25