


面试官：听说你sql写的挺溜的，你说一说查询sql的执行过程

程序员面试 5月10日

以下文章来源于非科班的科班，作者黎杜



非科班的科班

世界上并没有什么救世主，假如有那便是你自己；世界上也没有什么奇迹，假如有那只...



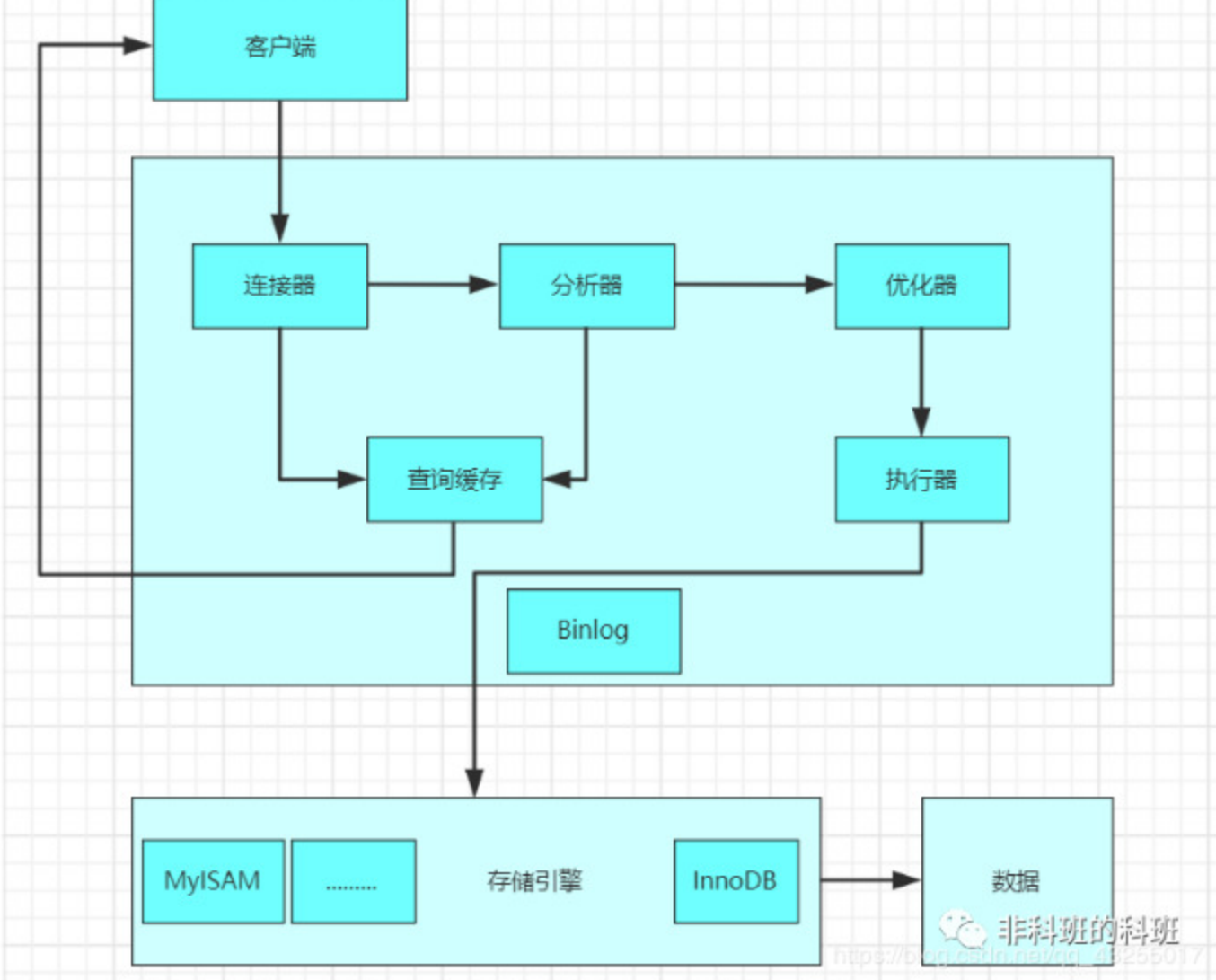


当希望Mysql能够高效的执行的时候，最好的办法就是清楚的了解Mysql是如何执行查询的，只有更加全面的了解SQL执行的每一个过程，才能更好的进行SQL的优化。

当执行一条查询的SQL的时候大概发生了一下的步骤：

1. 客户端发送查询语句给服务器。
2. 服务器首先检查缓存中是否存在该查询，若存在，返回缓存中存在的结果。若是不存在就进行下一步。
3. 服务器进行SQL的解析、语法检测和预处理，再由优化器生成对应的执行计划。
4. Mysql的执行器根据优化器生成的执行计划执行，调用存储引擎的接口进行查询。
5. 服务器将查询的结果返回客户端。

Mysql的执行的流程图如下图所示：



这里以一个实例进行说明Mysql的的执行过程，新建一个User表，如下：

```
// 新建一个表
DROP TABLE IF EXISTS User;
CREATE TABLE `User` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(10) DEFAULT NULL,
  `age` int DEFAULT 0,
  `address` varchar(255) DEFAULT NULL,
  `phone` varchar(255) DEFAULT NULL,
  `dept` int,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=40 DEFAULT CHARSET=utf8;

// 并初始化数据，如下
INSERT INTO User (name,age,address,phone,dept)VALUES('张三',24,'北京','13265543552',2);
INSERT INTO User (name,age,address,phone,dept)VALUES('张三三',20,'北京','13265543557',2);
INSERT INTO User (name,age,address,phone,dept)VALUES('李四',23,'上海','13265543553',2);
INSERT INTO User (name,age,address,phone,dept)VALUES('李四四',21,'上海','13265543556',2);
INSERT INTO User (name,age,address,phone,dept)VALUES('王五',27,'广州','13265543558',3);
INSERT INTO User (name,age,address,phone,dept)VALUES('王五五',26,'广州','13265543559',3);
INSERT INTO User (name,age,address,phone,dept)VALUES('赵六',25,'深圳','13265543550',3);
INSERT INTO User (name,age,address,phone,dept)VALUES('赵六六',28,'广州','13265543561',3);
INSERT INTO User (name,age,address,phone,dept)VALUES('七七',29,'广州','13265543562',4);
INSERT INTO User (name,age,address,phone,dept)VALUES('八八',23,'广州','13265543563',4);
INSERT INTO User (name,age,address,phone,dept)VALUES('九九',24,'广州','13265543564',4);
```

现在针对这个表发出一条SQL查询：**查询每个部门中25岁以下的员工个数大于3的员工个数和部门编号，并按照人工个数降序排序和部门编号升序排序的前两个部门。**

```
SELECT dept,COUNT(phone) AS num FROM User WHERE age< 25 GROUP BY dept HAVING num >= 3 ORDER BY num DESC,dept ASC LIMIT 2;
```

执行连接器

开始执行这条sql时，会检查该语句是否有权限，若是没有权限就直接返回错误信息，有权限会进行下一步，校验权限的这一步是在图一的连接器进行的，对连接用户权限的校验。

执行检索内存

相连建立之后，履行查询语句的时候，会先行检索内存，Mysql会先行冗余这个sql与否履行过，以此 **Key-Value** 的形式平缓适用内存中，Key是 **检索预定**，Value是 **结果集**。

假如内存key遭击中，便会间接回到给客户端，假如没命中，便会履行后续的操作，完工之后亦会将结果内存存上去，当下一次进行查询的时候也是如此的循环操作。

执行分析器

分析器主要有两步：（1）**词法分析** （2）**语法分析**

词法分析主要执行 **提炼关键字**，比如select，**提交检索的表**，**提交字段名**，**提交检索条件**。语法分析主要执行辨别你 **输出的sql**与否准确，是否 **合乎mysql**的语法。

当Mysql没有命中内存的时候，接着执行的是 FROM student 负责把数据库的表文件加载到内存中去，**WHERE age< 60**，会把所示表中的数据进行过滤，取出符合条件的记录行，生成一张临时表，如下图所示。

id	name	age	address	phone	dept
40	张三	24	北京	13265543552	2
41	张三三	20	北京	13265543557	2
42	李四	23	上海	13265543553	2
43	李四四	21	上海	13265543556	2
49	八八	23	广州	13265543563	4
50	九九	24	广州	13265543564	4

**GROUP BY dept** 会把上图的临时表分成若干临时表，切分的过程如下图所示：

id	name	age	address	phone	dept
40	张三	24	北京	13265543552	2
41	张三三	20	北京	13265543557	2
42	李四	23	上海	13265543553	2
43	李四四	21	上海	13265543556	2

id	name	age	address	phone	dept
49	八八	23	广州	13265543563	4
50	九九	24	广州	13265543564	4

查询的结果只有部门2和部门3才有符合条件的值，生成如上两图的临时表。接着执行 **SELECT 后面的字段**，SELECT后面可以是 **表字段** 也可以是 **聚合函数**。

这里SELECT的情况与是否存在 **GROUP BY** 有关，若是不存在Mysql直接按照上图内存中整列读取。若是存在分别SELECT临时表的数据。

最后生成的临时表如下图所示：

dept	num
2	4
4	2

紧接着执行 **HAVING num>2** 过滤员工数小于等于2的部门，对于 **WHERE** 和 **HAVING** 都是进行过滤，那么这两者有什么不同呢？

第一点是WHERE后面只能对表字段进行过滤，不能使用聚合函数，而HAVING可以过滤表字段也可以使用聚合函数进行过滤。

第二点是WHERE是对执行from User操作后，加载表数据到内存后，WHERE是对 **原生表的字段** 进行过滤，而HAVING是对 **SELECT后的字段进行过滤**，也就是WHERE **不能使用别名进行过滤**。

因为执行WHERE的时候，还没有SELECT，还没有给字段赋予别名。接着生成的临时表如下图所示：

dept	num
2	4

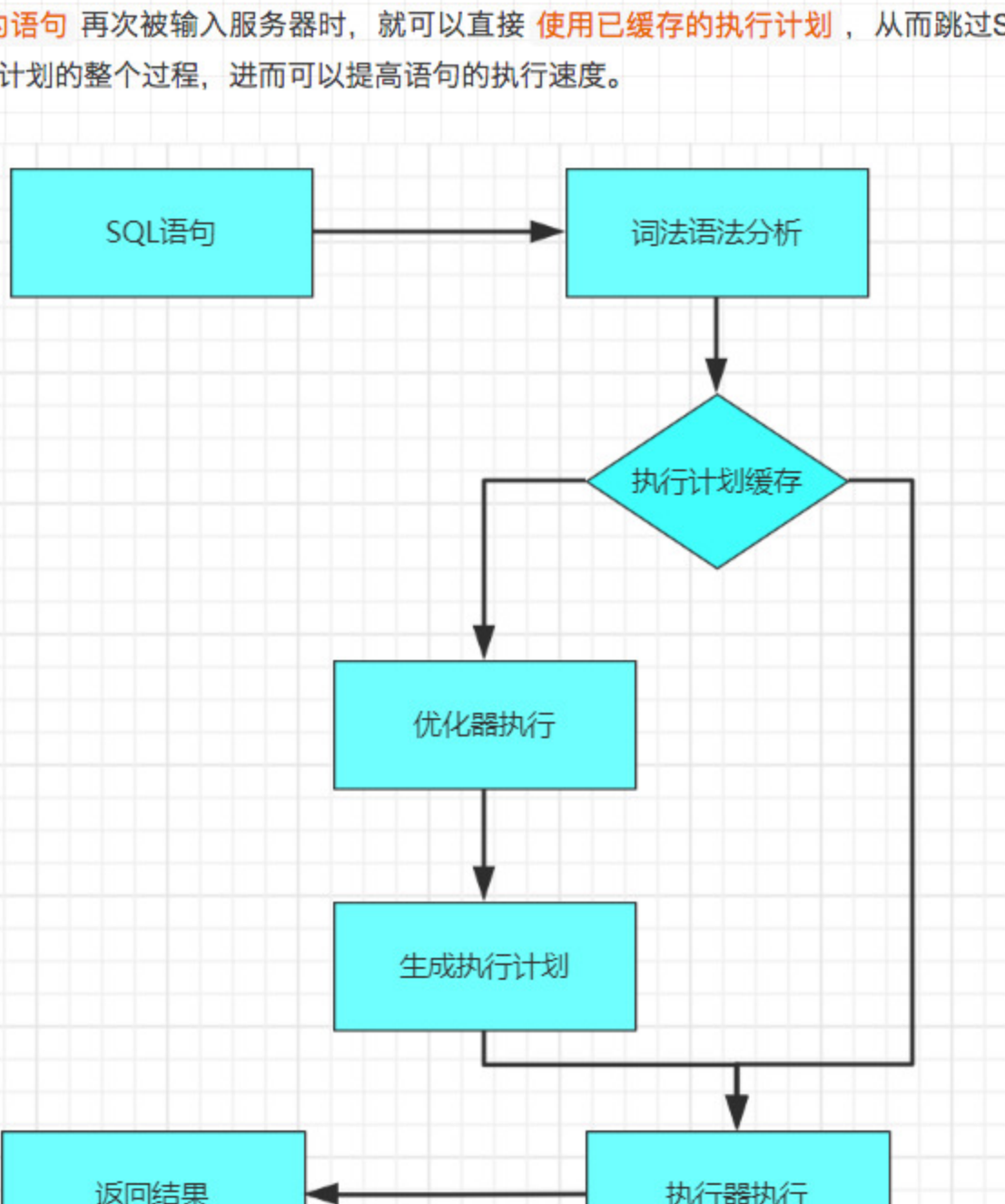
最后在执行 **ORDER BY后面的排序** 以及 **limit0,2** 取得前两个数据，因为这里数据比较少，没有体现出来。最后生成的结果也是如上图所示。接着判断这个sql语句 **是否有语法错误**，**关键字词与否准确** 等等。

执行优化器

查询优化器会将解析树转化成执行计划。一条查询可以有多种执行方法，最后都是返回相同结果。优化器的作用就是找到这其中 **最好的执行计划**。

生成执行计划的过程会消耗较多的时间，特别是存在许多可选的执行计划时。如果在一条SQL语句执行的过程中将该语句对应的最终执行计划进行缓存。

当 **相似的语句** 再次被输入服务器时，就可以直接 **使用已缓存的执行计划**，从而跳过SQL语句生成执行计划的整个过程，进而可以提高语句的执行速度。



MySQL使用基于成本的查询优化器。它会尝试预测一个查询使用某种执行计划时的成本，并选择其中成本最少的一个。

执行执行器

由优化器生成得执行计划，交由执行器进行执行，执行器调用存储引擎得接口，存储引擎获取数据并返回，结束整个查询得过程。

这里之讲解了select的过程，对于update这些修改数据或者删除数据的操作，会涉及到事务，会使用两个日志模块，redo log和binlog日志。

以前的Mysql的默认存储引擎MyISAM引擎是没redo log的，而现在的默认存储引擎InnoDB引擎便是透过redo 复杂度来拥护事务的，保证事务能够准确的回滚或者提交，保证事务的ACID。

长按订阅更多面经分享



微信扫一扫  
关注该公众号