分析一个sql语句在MySQL中的执行流程：

* 包括sql的查询MySQL内部会怎么流转
* sql语句的更新是怎么完成的

先了解MySQL的基础架构，知道MySQL由哪些组件组成以及这些组件的作用是什么。

### 1、MySQL基础架构分析



基础结构组成：

* 连接器：身份认证和权限相关（登录MySQL的时候）
* 查询缓存：执行查询语句的时候，会先查询缓存（MySQL 8.0版本后移除）
* 分析器：没有命中缓存的话，SQL语句就会经过分析器，分析器就是要先看你的SQL语句要干嘛，再检查SQL语句语法是否正确。
* 优化器：按照MySQL认为最优的方案去执行。
* 执行器：执行语句，然后从存储引擎返回数据。

MySQL主要分为Server层和存储引擎层：

* Server层：主要包括连接器、查询缓存、分析器、优化器、执行器等，所有跨存储引擎的功能都在这一层实现，比如存储过程、触发器、视图、函数等，还有一个通用的日志模块binlog日志模块。
* 存储引擎层：主要负责数据的存储和读取，采用可以替换的插件式架构，支持InnoDB、MyISAM、Memory等多个存储引擎，其中InnoDB引擎有自有的日志模块redolog模块。

#### 1) 连接器

连接器主要和身份认证和权限相关的功能相关，就好比一个级别很高的门卫一样。

主要负责用户登录数据库，进行用户的身份认证，包括校验账户密码，权限等操作，如果用户账户密码已通过，连接器会到权限表中查询该用户的所有权限，之后在这个连接里的权限逻辑判断都是会依赖此时读取到的权限数据，

注：后续只要这个连接不断开，即时管理员修改了该用户的权限，该用户也是

不受影响的。

#### 2) 查询缓存(MySQL 8.0 版本后移除)

查询缓存主要用来缓存我们所执行的 SELECT 语句以及该语句的结果集。

连接建立后，执行查询语句的时候，会先查询缓存，MySQL会先校验这个sql是否执行过，以Key-Value的形式缓存在内存中，Key是查询预计，Value是结果集。如果缓存key被命中，就会直接返回给客户端，如果没有命中，就会执行后续的操作，完成后也会把结果缓存起来，方便下一次调用。当然在真正执行缓存查询的时候还是会校验用户的权限，是否有该表的查询条件。

MySQL查询不建议使用缓存，因为查询缓存失效在实际业务场景中可能会非常频繁，假如你对一个表更新的话，这个表上的所有的查询缓存都会被清空。对于不经常更新的数据来说，使用缓存还是可以的。MySQL 8.0 版本后删除了缓存的功能

#### 3) 分析器

MySQL没有命中缓存，就会进入分析器，分析器主要是用来分析SQL语句是来干嘛的，分析器也会分为几步：

1. 词法分析，一条SQL语句有多个字符串组成，首先要提取关键字，比如select，提出查询的表，提出字段名，提出查询条件等等。做完这些操作后，就会进入第二步。
2. 语法分析，主要就是判断你输入的sql是否正确，是否符合MySQL的语法。
3. 完成上2步之后，MySQL就准备开始执行了，但是如何执行，怎么执行是最好的结果呢？这个时候就需要优化器上场了。

#### 4) 优化器

优化器的作用就是它认为的最优的执行方案去执行（有时候可能也不是最优），比如多个索引的时候该如何选择索引，多表查询的时候如何选择关联顺序。

可以说，经过了优化器之后可以说这个语句具体该如何执行就已经定下来。

#### 5) 执行器

当选择了执行方案后，MySQL就准备开始执行了，首先执行前会校验该用户有没有权限，如果没有权限，就会返回错误信息，如果有权限，就会去调用引擎的接口，返回接口执行的结果。

### 2、语句分析

#### 2.1 查询语句

SELECT

\*

FROM

USER A

WHERE

A.age = 18

AND A.`name` = 'Jone';

执行流程：

* 先检查该语句是否有权限，如果没有，直接返回错误信息，如果有权限，在MySQL 8.0以前会先查询缓存，以这条sql语句为key在内存中查询是否有结果，如果有直接返回。没有则执行下下一步。
* 通过分析器进行词法分析，提取sql的关键元素，比如提取这个语句是查询select，提取需要查询的表名为user，需要查询所有列，查询条件是这个表的age=18，‘name’=‘Jone’。然后判断这个sql语句是否有语法错误，比如关键字是否正确等等，如果检查没问题就执行下一步。
* 接下来就是优化器进行确定执行方案，上面的sql语句可以有2种执行方案：
  + 先查询用户表中姓名为“Jone”的用户，然后判断是否年龄为18；
  + 先找出年龄为18的学生，然后再查询姓名为“Jone”的学生。
* 进行权限校验，如果没有权限就会返回错误信息，如果有权限就会调用数据库引擎接口，返回引擎的执行结果。

#### 2.2 更新语句（增删改）

UPDATE `user` A

SET A.age = '19'

WHERE

A.`name` = 'Jone';

这条sql基本也会沿着查询语句的流程走，只不过执行更新的时候肯定要记录日志，这就会引入日志模块，MySQL自带的日志模块binlog（归档日志），所有的存储引擎都可以使用，InnoDB引擎还自带一个日志模块redo log（重做日志），以InnoDB引擎为例，流程如下：

* 先查询到Jone这一条数据，如果有缓存，也是会用到缓存。
* 然后拿到更新的语句，把age改为19，然后调用引擎接口，写入这一行数据，InnoDB引擎把数据保存在内存中，同时记录redo log日志，此时redo log进入prepare状态，然后告诉执行器，执行完成了，随时可以提交。
* 执行器收到通知后记录binlog，然后调用引擎接口，提交redo log为提交状态。
* 更新完成。

### 3、为什么要用两个日志模块

这是因为最开始MySQL并没有InnoDB引擎(InnoDB引擎是其他公司以插件形式插入MySQL的) ，redo log是InnoDB引擎特有的，这就导致会没有 crash-safe的能力(即使数据库发生异常重启，之前提交的记录都不会丢失)，binlog 日志只能用来归档。

并不是说只用一个日志模块不可以，只是InnoDB引擎就是通过redo log来支持事务的。用两个日志模块，为什么redo log要引入prepare预提交状态？这里我们用反证法来说明下为什么要这么做？

* 先写redo log直接提交，然后写 binlog，假设写完 redo log 后，机器挂了，binlog 日志没有被写入，那么机器重启后，这台机器会通过 redo log 恢复数据，但是这个时候 binlog 并没有记录该数据，后续进行机器备份的时候，就会丢失这一条数据，同时主从同步也会丢失这一条数据。
* 先写binlog，然后写redo log，假设写完了binlog，机器异常重启了，由于没有redo log，本机是无法恢复这一条记录的，但是binlog又有记录，那么和上面同样的道理，就会产生数据不一致的情况。
* 如果采用redo log两阶段提交的方式就不一样了，写完binglog后，然后再提交redo log就会防止出现上述的问题，从而保证了数据的一致性。那么问题来了，有没有一个极端的情况呢？假设redo log处于预提交状态，binglog也已经写完了，这个时候发生了异常重启会怎么样呢？这个就要依赖于MySQL的处理机制了，MySQL 的处理过程如下：
  + 判断 redo log 是否完整，如果判断是完整的，就立即提交。
  + 如果 redo log 只是预提交但不是 commit 状态，这个时候就会去判断 binlog 是否完整，如果完整就提交 redo log, 不完整就回滚事务。