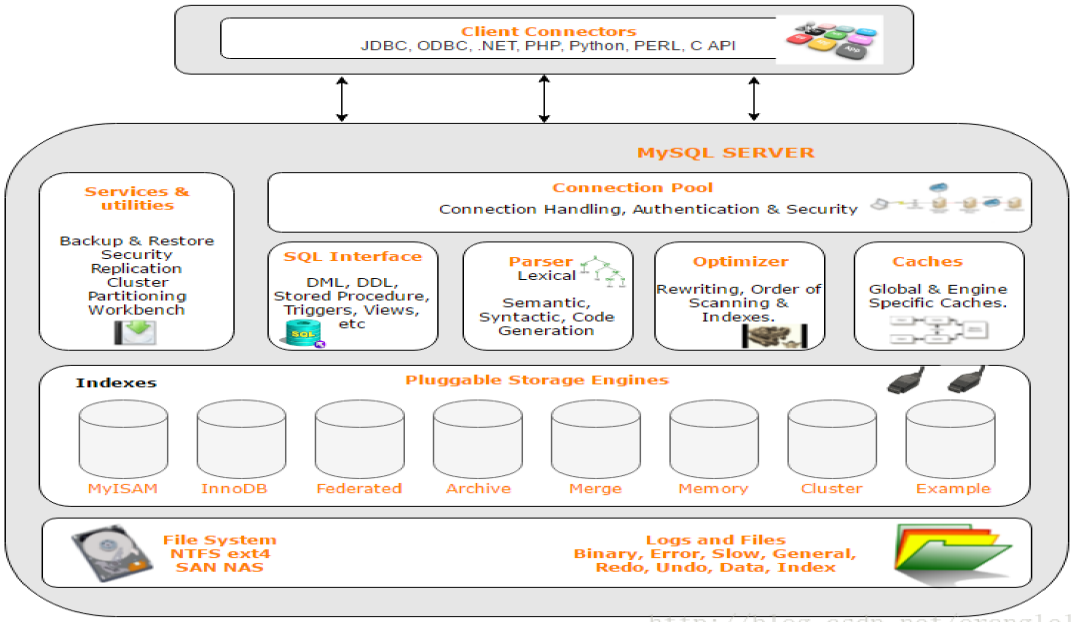
MySQL训练营-架构详解及SQL执行流程

本内容属于动脑学院Steven老师首创,如需转载,请注明出处!

Steven QQ : 750954883

动脑学院系列课程咨询: QQ 2729772006

MySQL体系结构的再认识



MySQL组件名称	用途介绍
Client Connectors	指接入方，MySQL服务端的请求客户端
Management Serveices & Utilities	系统管理和控制工具，mysqldump、 mysql复制集群、分区管理等
Connection Pool	连接池：管理及缓冲用户连接、用户名、密码、权限校验、线程处理等需要缓存的需求
SQL Interface	SQL接入口，接受用户的SQL命令，并且返回用户需要查询的结果
Parser	解析器，SQL命令传递到解析器的时候会被解析器验证和解析
Optimizer	查询优化器，SQL语句在查询之前会使用查询优化器对查询进行优化，并确立唯一的执行计划
Cache、Buffer	查询缓存，如果查询缓存有命中的查询结果，查询语句就可以直接去查询缓存中取数
Pluggable storage Engines	插件式存储引擎。存储引擎是MySQL中具体的与文件打交道的子系统
File system	文件系统，数据、日志（redo，undo，binlog）、索引、错误日志、查询记录、慢查询等

MySQL中各大存储引擎

官网介绍. https://dev.mysql.com/doc/refman/5.7/en/storage-engines.html

CSV

特点:

- 不能定义没有索引、列定义必须为NOT NULL、不能设置自增列
- CSV表的数据的存储用 “,” 隔开，可直接编辑CSV文件进行数据的编排
- 不适用大表或者数据的在线处理
- 数据安全性低

应用场景:

- 数据的快速导出导入
- 表格直接转换成CSV

我们可以直接通过文本工具打开CSV的数据表文件进行内容编排(满足规范),通过flush table XXX;即可完成数据的编辑

Archive

特点:

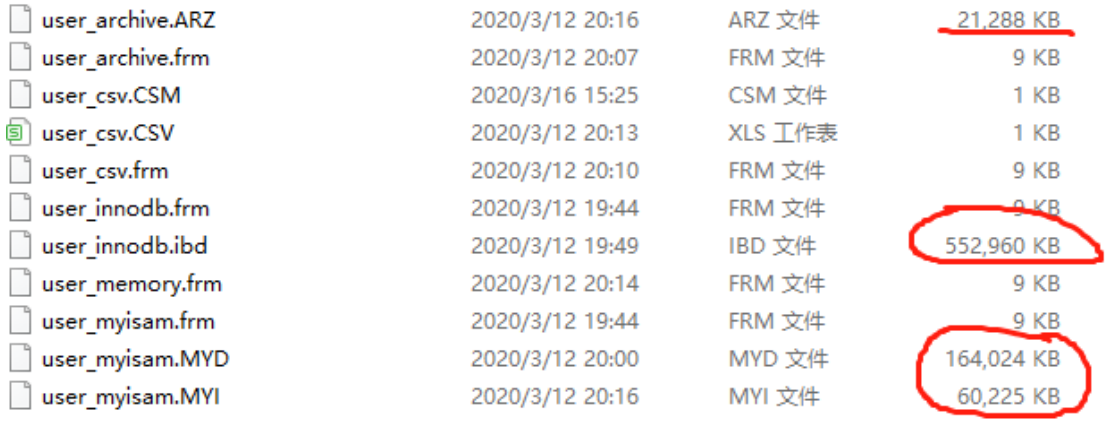
- 压缩协议(ARZ文件格式)进行数据的存储,磁盘占用少

- 只支持insert和select两种操作
- 只允许自增ID列建立索引

应用场景:

- 日志采集系统
- 大量设备数据采集及归档

相同的数据字段和数据行数,在Myisam引擎和Innodb引擎中所占用的大小将大的多.



user_archive.ARZ	2020/3/12 20:16	ARZ 文件	21,288 KB
user_archive.frm	2020/3/12 20:07	FRM 文件	9 KB
user_csv.CSM	2020/3/16 15:25	CSM 文件	1 KB
user_csv.CSV	2020/3/12 20:13	XLS 工作表	1 KB
user_csv.frm	2020/3/12 20:10	FRM 文件	9 KB
user_innodb.frm	2020/3/12 19:44	FRM 文件	9 KB
user_innodb.ibd	2020/3/12 19:49	IBD 文件	552,960 KB
user_memory.frm	2020/3/12 20:14	FRM 文件	9 KB
user_myisam.frm	2020/3/12 19:44	FRM 文件	9 KB
user_myisam.MYD	2020/3/12 20:00	MYD 文件	164,024 KB
user_myisam.MYI	2020/3/12 20:16	MYI 文件	60,225 KB

Memory

https://dev.mysql.com/doc/refman/5.7/en/memory-storage-engine.html#memory-storage-engine-characteristics-of-memory-tables

特点:

- 数据都是存储在内存中，处理效率高，表大小限定默认16M
- 不支持大数据存储类型的字段 如 blog，text
- 支持Hash索引，等值查询效率高
- 字符串类型强制使用char 如varchar(32) --> char(32)
- 数据的可靠性低，重启或系统奔溃数据丢失

应用场景:

- 数据热点快速加载
- 查询结果内存中的计算，大多数都是采用这种存储引擎作为临时表存储需计算的数据

Myisam

特点:

- 较快的数据插入和读取性能
- 较小的磁盘占用[相较于Innodb]
- 支持表级别的锁，不支持事务
- 数据文件与索引文件分开存储[MYD和MYI]

应用场景:

- 只读应用或者以读为主的业务

MySQL5.1版本之前默认存储引擎是Myisam

Innodb

特点:

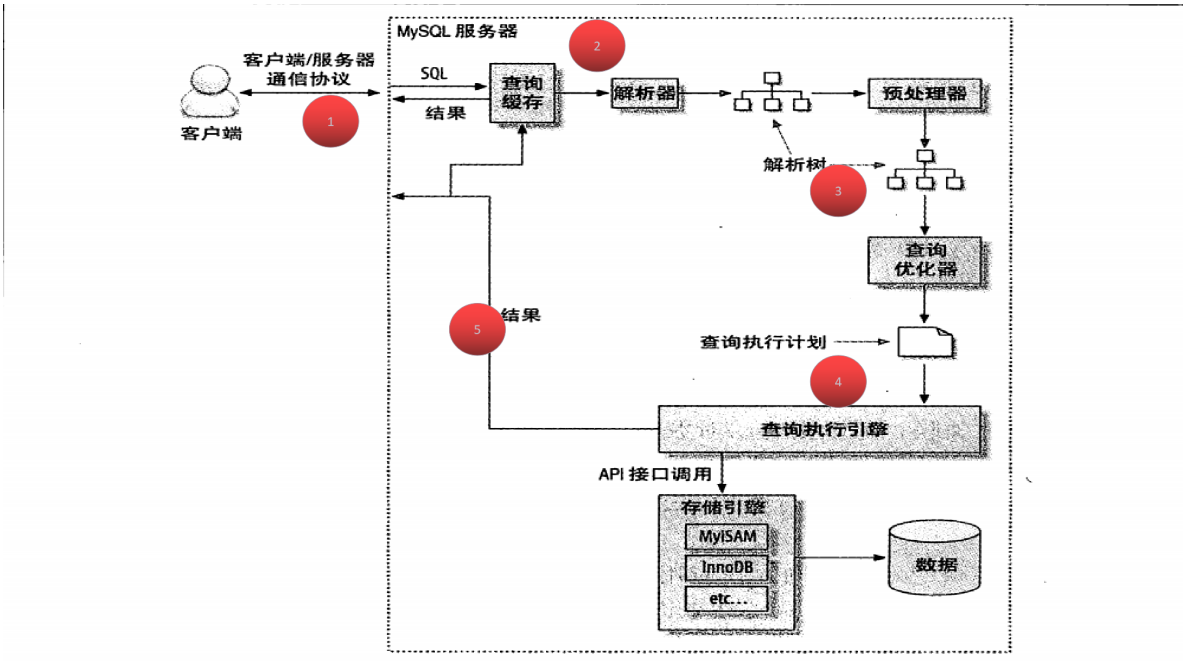
- 支持事务
- 行级锁
- 聚集索引
- 数据行内容与索引结构在一个文件[IBD]
- 外键支持，保证数据的完整性

应用场景:

- 无脑选择

MySQL5.5及版本及之后的MySQL默认存储引擎

一条SQL执行的五个流程



我们把SQL执行的整个流程分为5个阶段.分别为

- C-S的通讯阶段
- 查询缓存阶段
- SQL验证和优化阶段
- 执行引擎执行阶段
- 数据返回阶段

step1-客户端服务端通讯

协议选择

MySQL 客户端与服务端的连接协议主要包括两种

- TCP/IP

JDBC,ODBC,.NET,Python....

- Unix socket

```
Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)
```

通讯方式

MySQL通讯采用**半双工**长连接方式

- 全双工(打电话)

  指在发送数据的同时也能够接收数据，两者同步进行，这好像我们平时打电话一样，说话的同时也能够听到对方的声音

- 半双工(对讲机)

  数据的传输是双向的,但是同一时间内，只有一种动作发生,一方正在发送数据另外一方就只能等待数据传输完毕才能发起数据传输动作.

- 单工(收音机)

  指数据的传输按单一的方向的进行传输.

半双工连接方式的特点

- 同一时间点只有一方向另一方发送数据
- 一旦发送数据必须等数据发送完成才能继续

启示

- MySQL服务端限制接收的字符串大小 `show variables like 'max_allowed_packet'` 默认值为4M

  我们可以通过调整这个值的大小进行配置

- 针对SQL的发送,我们需求多少条数据就查询多少条数据.针对SQL语句一定要记得加上`Limit`关键字.

  我们在使用sqlyog后者navicat等工具的时候,我们往往很任性,但是在SQL编码的时候,我们的SQL语句就是我 们发出的语句,所以一定要记得使用`Limit`关键字进行需求数据量的返回

思考题

一个查询SQL语句发出去之后很久没有回应怎么办?

如果是在linux环境我们可以查看MySQL的服务端口情况.

`netstat -an|grep 3306 -wc`

或者使用 `show full processlist` 查看MySQL连接数的状态.

```
mysql> show full processlist;
+----+------+-----------------+---------+---------+------+----------+----------------------+
| Id | User | Host            | db      | Command | Time | State    | Info                 |
+----+------+-----------------+---------+---------+------+----------+----------------------+
| 17 | root | localhost:59217 | b-tree  | Sleep   | 5413 |          | NULL                 |
| 18 | root | localhost:59280 | b-tree  | Sleep   | 7352 |          | NULL                 |
| 20 | root | localhost:60424 | NULL    | Sleep   | 5414 |          | NULL                 |
| 21 | root | localhost:60425 | example | Sleep   | 3924 |          | NULL                 |
| 22 | root | localhost:60428 | example | Sleep   | 5402 |          | NULL                 |
| 23 | root | localhost:60429 | example | Sleep   | 5397 |          | NULL                 |
| 24 | root | localhost:60442 | example | Sleep   | 5102 |          | NULL                 |
| 25 | root | localhost:60445 | example | Sleep   | 3700 |          | NULL                 |
| 27 | root | localhost:62401 | NULL    | Query   |    0 | starting | show full processlist |
+----+------+-----------------+---------+---------+------+----------+----------------------+
9 rows in set (0.00 sec)
```

具体每一种状态代表的含义可通过官方文档进行查询-> https://dev.mysql.com/doc/refman/5.7/en/general-thread-states.html

我们可找到我们发出的SQL.强制中断连接 通过 `kill {id}` 指令

数据连接报出Too many connections ?

show status like 'Thread%'

```
mysql> show status like 'Threads%';
+-------------------+--------+
| Variable_name     | Value  |
+-------------------+--------+
| Threads_cached    | 0      |
| Threads_connected | 9      |
| Threads_created   | 9      |
| Threads_running   | 1      |
+-------------------+--------+
4 rows in set (0.00 sec)
```

- Threads_cached 代表目前还有多少连接可以复用.
- Threads_connected 代表目前连接到MySQL服务器的连接数
- Threads_created 代表一定的时间内连接的创建量.

  若你的程序没做一次查询就创建一个connection对象,有可能这个值就会很大.
- 当下正在执行命令的连接.

所以可以基于分析.也可以基于实际的连接情况可以适时的调整连接数的大小.

`show variables like 'max_conections'` 查看目前MySQL数据库能承载的最大连接数.

`set global max_connections = 1000;` **这个重启服务就会失效.**

或者在my.cnf 文件中进行最大连接数的修改.如果在linux系统的话,需要考虑open file limit.具体官方文档介绍

https://dev.mysql.com/doc/refman/5.7/en/client-connections.html

step2-查询缓存

查询缓存功能和原理

在互联网初期阶段人们利用互联网只是单纯的改变浏览阅读习惯和少量的知识内容的获取.所以很多的应用场景只存在于内容的展示和单纯的阅读.并不像现在,我们的MySQL的优势在于OLTP系统的数据生产.所以针对读多写少的应用场景.MySQL开发了查询缓存的功能.

此项功能在5.6版本就被MySQL自动关闭,且在MySQL8.0版本直接摒弃.但是这个功能在SQL执行的流程中确是值得关注的一个重要环节.

查询缓存的工作原理

- 缓存SELECT操作的结果集和SQL语句;
- 基于接收的SELECT语句，先去查询缓存，判断是否存在可用的记录集;

限制条件:

- 缓存失效与表数据是一致的,即表数据发生变化就缓存失效
- 需要手工开启. 5.6,5.7 都需要手工开启 /大多数的应用场景并不适用
- 使用/判断条件苛刻 SQL语句必须完全一致,中间加上一个空格都是缓存命中不到.
- 带来额外的性能消耗

查询缓存配置

`query_cache_type` 值：0 -- 不启用查询缓存，默认值； 值：1 -- 启用查询缓存，只要符合查询缓存的要求，客户端的查询语句和记录集都可以缓存起来，供其他客户端使用，加上 SQL_NO_CACHE将不缓存 值：2 -- 启用查询缓存，只要查询语句中添加了参数：SQL_CACHE，且符合查询缓存的要求，客户端的查询语句和记录集，则可以缓存起来，供其他客户端使用

`query_cache_size` 允许设置query_cache_size的值最小为40K，默认1M，推荐设置 为：64M/128M;

`query_cache_limit` 限制查询缓存区最大能缓存的查询记录集，默认设置为1M

`show status like 'Qcache%'` 命令可查看缓存情况

step3-验证优化

验证阶段

Parser会基于发送过来的SQL进行语法分析,词法分析得到一颗对应SQL的解析树.

语法分析:验证SQL语句的语法正确性,如 恶意的字符串 ()没有正确结束, 引号没有正确结束 返回1064

词法分析:将整条SQL语句解析成一颗解析树,基于此树结构进行表名,字段名,函数,权限等等相关的验证.

优化阶段

基于验证阶段Parser组件生成的解析树.Optimizer(查询优化器)将通过物理和逻辑两层优化找到最优的执行计划.

物理优化过程

- 使用等价变化规则

5 = 5 and a > 5 改写成 a > 5 a < b and a = 5 改写成 b > 5 and a = 5

- 基于联合索引，调整条件位置等

  index(a,b,c) SQL where a =xx and c =ddd

  EX:

  explain select * from customer where last_name = 'ANDERSON' and first_name = 'LISA';

逻辑优化过程

- 优化count,min,max等相关函数

  索引和列是否可为空通常可以帮助 MySQL 优化这类表达式。例如，要找到某一列的最小值，只需要查询对应 B-Tree 索引最左端的记录，MySQL 可以直接获取索引的第一行记录。在优化器生成执行计划的时候就可以利用这一点，在 B-Tree 索引中，优化器会将这个表达式作为一个常数对待。类似的，如果要查找一个最大值，也只需读取 B-Tree 索引的最后一条记录。如果 MySQL 使用了这种类型的优化，那么在 EXPLAIN 中就可以看到“Select tables optimized away”。从字面意思可以看出，它表示优化器已经从执行计划中移除了该表，并以一个常数取而代之。
  类似的，没有任何 WHERE 条件的 COUNT(*) 查询通常也可以使用存储引擎提供的一些优化（例如，MyISAM 维护了一个变量来存放数据表的行数）。

  min优化基于B+Tree的索引,只需要查询对应的索引叶子节点的最左端记录即可.

  max优化基于B+Tree的索引,只需要查询对应的索引叶子节点的最右端记录即可.

  `explain select min(customer_id) from customer;`

  count函数,在Myisam引擎中,表记录行数是以常量的方式保存的.针对Myisam引擎的count查询,只需要查询该常量即可.

  `explain select count(*) from user_myisam;`

  通过Explain这些类型的SQL语句,会发现,在Extra的列中显示 `Select tables optimized away`.

  表示 优化器针对这个SQL的优化采用常数取而代之

- 覆盖索引扫描优化

  如果查询语句的列字段可以通过索引信息直接返回,则无需回表进行二次查询.

  `explain select name from user_innodb where name like 'hello-123456%' ;`

  ```
  mysql> explain select name from user_innodb where `name` like 'hello-123456%' \G
  *************************** 1. row ***************************
             id: 1
    select_type: SIMPLE
          table: user_innodb
     partitions: NULL
           type: range
  possible_keys: idx_name
            key: idx_name
        key_len: 131
            ref: NULL
           rows: 11
       filtered: 100.00
          Extra: Using where; Using index
  1 row in set, 1 warning (0.00 sec)
  ```

  在对应的Extra信息我们发现有Using index 的额外信息,表示该SQL采用索引进行数据的返回.

- 冗余子查询的优化

`explain select * from (select * from user_innodb where id =1111) as T`

```
mysql> explain select * from (select * from user_innodb where id =1111) as T\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: user_innodb
   partitions: NULL
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
1 row in set, 1 warning (0.00 sec)
```

- 提前终止查询

  在发现已经满足查询需求时,MySQL能够立刻终止查询.

  这种立刻终止查询主要体现在两种场景:

  - limit子句,查询10条数据,一旦发现满足了10条就立刻终止查询
  - 第二种场景逻辑上不成立的查询条件.若 id列最小值为1 where id =-1 这样的条件.

```
mysql> explain select * from user_innodb where id = -1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: no matching row in const table
1 row in set, 1 warning (0.00 sec)
```

`explain select * from user_innodb where id = -1 \G`

上图中Extra中提示`no matching row in const table`

- in的条件优化

列表 IN() 的比较

在很多数据库系统中，IN() 完全等同于多个 OR 条件的子句，因为这两者是完全等价的。在 MySQL 中这点是不成立的，MySQL 将 IN() 列表中的数据先进行排序，然后通过二分查找的方式来确定列表中的值是否满足条件，这是一个 O(log n) 复杂度的操作，等价地转换成 OR 查询的复杂度为 O(n)，对于 IN() 列表中有大量取值的时候，MySQL 的处理速度将会更快。

总而言之,在优化器的逻辑优化阶段,MySQL的查询优化器是基于成本计算的原则。他会去尝试各种执行计划。并进行抽样数据的方式进行试验. 优化器的抽样:采用随机的读取一个4K的数据块进行分析。

执行计划

基于上面的验证和优化2个步骤之后.整个step3阶段最终要得到一个parser与optimizer介入之后的最优执行计划.

执行计划指的是针对一条SQL最终在MySQL数据库执行的时候,SQL执行过程中的步骤.以及每一个步骤的具体操作.

了解SQL执行计划的意义就在于我们可以通过执行计划更加清晰的认识到这一条语句，分为了哪几步，有没有用到索引，是否有一些可优化的地方

如何得到执行计划

- 记录执行计划到系统表 **information_schema.optimizer_trace**

  `show variables like 'optimizer_trace'` 查看是否开启记录执行计划生成日志.

  `set global optimizer_trace= 'enabled=on'` 打开全局的执行计划日志

- 通过explain / desc + SQL的方式得到该SQL语句的最后执行计划.

执行计划详解

ex:`explain select * from exp_user where userID = (select userID from exp_user_address WHERE addrID = 1)\G`

```
mysql> explain  select * from exp_user where userID  = (select userID  from exp_user_address WHERE addrID =  1)\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: exp_user
   partitions: NULL
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           id: 2
  select_type: SUBQUERY
        table: exp_user_address
   partitions: NULL
         type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (0.00 sec)
```

`explain select * from exp_user where userID in (select userID from exp_user_address WHERE addrDetail like "北京%")\G`

```
mysql> explain select * from exp_user where userID  in (select userID  from exp_user_address WHERE addrDetail  like "北
京%")\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: exp_user
   partitions: NULL
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: exp_user_address
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
     filtered: 10.00
        Extra: Using where; FirstMatch(exp_user); Using join buffer (Block Nested Loop)
2 rows in set, 1 warning (0.00 sec)
```

执行计划详细官方详细描述:

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html

id

select查询的序列号，包含一组数字，表示查询中执行select子句或操作表的顺序

1、id相同：执行顺序由上至下

2、id不同：如果是子查询，id的序号会递增，id值越大优先级越高，越先被执行

3、id相同又不同（两种情况同时存在）：id如果相同，可以认为是一组，从上往下顺序执行；在所有组中，id值越大，优先级越高，越先执行

select_type

查询的类型，主要是用于区分普通查询、联合查询、子查询等复杂的查询

常见的取值

- Simple 简单查询(普通查询)
- PRIMARY 最外层查询
- SUBQUERY 子查询
- UNION 连接查询
- UNION RESULT 连接查询的结果集

具体的取值分析,可以打开官网进行查询

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_select_type

table

查询涉及到的表或者表的别名

常见取值

- 表名或者取的别名

- <union*M*,*N*> 查询id为M和N2个结果集进行并集结果
- <derived*N*> 派生表查询id为N的结果集

  派生表是从select语句返回的虚拟表。派生表类似于临时表，但是在SELECT语句中使用派生表比临时表简单得多，因为它不需要创建临时表的步骤。所以当SELECT语句的FROM子句中使用独立子查询时，我们将其称为派生表
- <subquery*N*> id为N子查询的结果列表

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain_table

partitions

查询将从中匹配记录的分区。该值适用`NULL`于未分区的表

type

数据获取的方式.

SQL查询优化中一个很重要的指标,他是评测SQL好坏最直观的参数.

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain-join-types

`system > const > eq_ref > ref > range > index > ALL` 一般来说，好的sql查询至少达到range级别，最好能达到ref

- system

  表只有一行记录（等于系统表），这是const类型的特例，平时不会出现，可以忽略不计.
- const

  表示基于唯一性索引(主键或者唯一索引)**唯一性条件为常数**的比较.

  ```
  explain select * from exp_user where userID = 1 \G
  ```
- eq_ref

  唯一性(主键,唯一)索引的扫描.

  ```
  explain select * from exp_user where userID in (select userID from
  exp_user_address where addrID in (1,6)) \G
  ```
- ref

  普通索引的扫描

  ```
  explain select * from exp_user where phoneNum = '13666666666' \G
  ```
- range

  扫描给定索引的范围行,一般在where语句中出现了bettween、<、>、in等的查询。这种索引列上的范围扫描比全索引扫描要好,只需要开始于某一行，结束于某一行，不用扫描全部索引

  ```
  explain select * from exp_user where phoneNum < '13666666666' \G
  ```

- index

  Full index Scan , 全索引扫描,这种类型只需要扫描索引树.（Index与ALL虽然都是读全表，但index是从索引中读取，而ALL是从表中读取）

  ```
  explain select userName from exp_user \G
  ```

- all

  Full Table Scan, 全表扫描,匹配数据行

  ```
  explain select * from exp_user \G
  ```

possible key

可能使用到的索引

key

真正使用到的索引.如果取值为NULL.则没有使用索引

key_len

标识使用到的索引列长度.

计算公式:

- varchr(10) 变长字段且允许NULL = 10 * (utfmb4=3 utf8=3,gbk=2,latin1=1)+1(NULL)+2(变长字段)
- varchr(10)变长字段且不允许NULL = 10 *(utfmb4=3 utf8=3,gbk=2,latin1=1)+2(变长字段)
- char(10)固定字段且允许NULL = 10 * (utfmb4=4 utf8=3,gbk=2,latin1=1)+1(NULL)
- char(10)固定字段且不允许NULL = 10 * (utfmb4=4 utf8=3,gbk=2,latin1=1)

```
explain select * from exp_user where phoneNum = '13666666666' and userName =
'Steven' \G
```

上诉的判断公式,是判断一个联合索引使用列的重要依据.

rows

根据表统计信息及索引选用情况，大致估算出找到所需的记录所需要读取的行数

filtered

它指返回结果的行占需要读到的行(rows列的值)的百分比

表示返回结果的行数占需读取行数的百分比，Filter列的值越大越好,表示扫描数据的准确性很高.

比如 rows为1000,filtered值为50 则真正返回的数据为500行.

Extra

- Using filesort

  MySQL对数据使用一个外部的文件内容进行了排序，而不是按照表内的索引进行排序读取

  也就是说MySQL无法利用已有索引完成的排序操作所以使用文件排序方式

  ```
  explain select * from exp_user order by lastUpdate desc\G   ---->使用文件排序
  explain select * from exp_user order by userID desc\G       ---->使用索引扫描
  ```

- Using temporary

  使用临时表保存中间结果，也就是说mysql在对查询结果排序时使用了临时表，常见于 group by 操作

  ```
  explain select max(lastUpdate) from exp_user GROUP BY lastUpdate \G
  ```

  若此时建立lastUpdate索引,则上诉情况消失.

- Using index

  表示相应的select操作中使用了覆盖索引（Covering Index），避免了访问表的数据行，效率高

- Using where

  使用了where 后面的条件进行过滤

- select tables optimized away

  基于索引优化MIN/MAX操作或者MyISAM存储引擎优化COUNT（*）操作，不必等到执行阶段在进行计算，查询执行计划生成的阶段即可完成优化

https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#explain-extra-information 具体其他的参数和情况可参见官方文档进行查阅

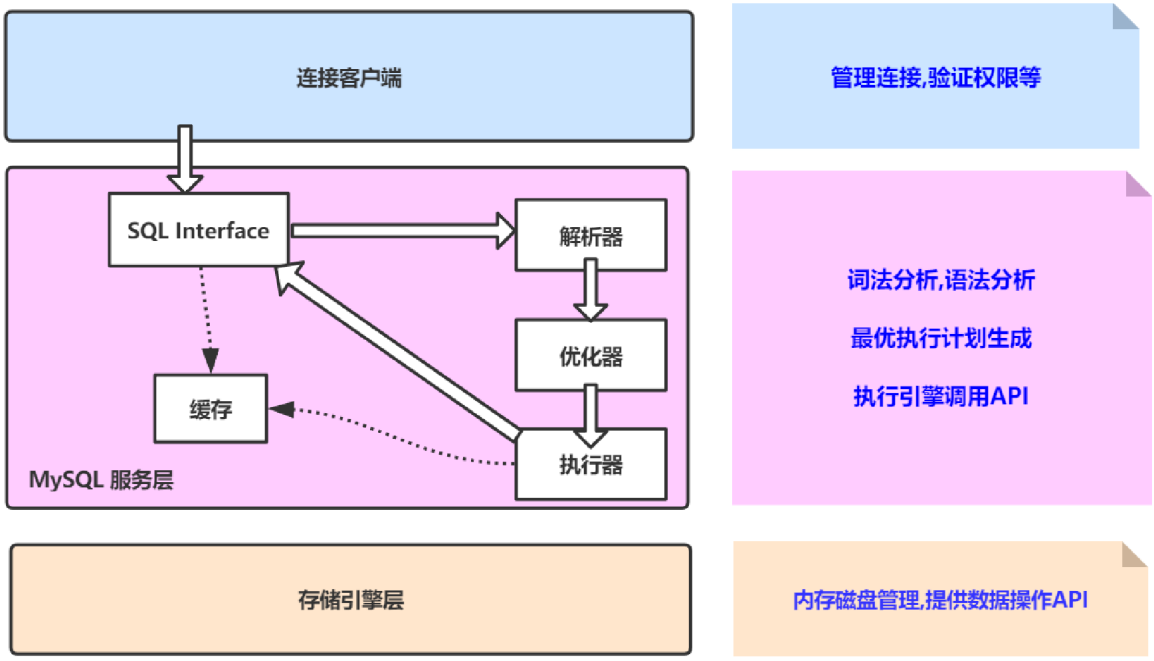
step4-执行引擎执行

调用API完成调用

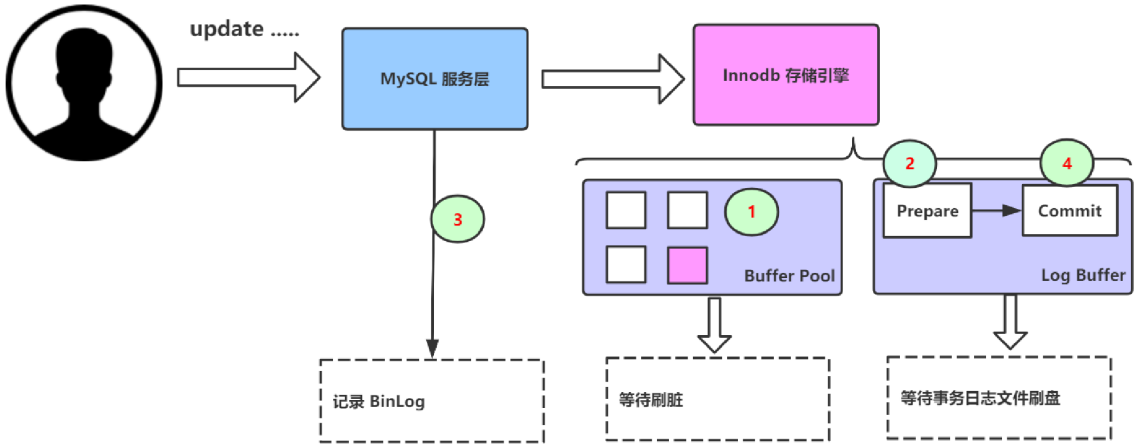
step5-数据返回

- 如果配置了查询缓存,则先执行缓存保存操作
- 进行数据的增量返回
  - 生成第一条记录时就开始返回数据
  - MySQL不保存返回的数据结果,不需要浪费额外的内存
  - 请求端数据响应快,立马可以得到数据结果

SQL执行标准流程及分层



DML语句的执行流程



通过XA的二阶段提交进行DML控制.

BinLog

BinLog是MySQL Server层一个日志文件.

BinLog以事件的形式记录了所有的DDL和DML语句操作.由于只记录操作的过程,并不是记录具体的数据变化所以BinLog属于逻辑日志.

BinLog以文件追加的方式进行记录,没有大小的限制.

BinLog是主从复制和数据恢复的神器.

BinLog记录方式

- Row 行的记录方式,如果是批量的修改,记录每一行记录变化SQL. 优势:记录每一行数据被修改的细节.不会因为函数,存储过程等带来数据不一致的问题 缺点:数据记录量大.
- Statement 默认 以事务为准,记录操作的本身SQL语句. 优势:减少数据记录. 缺点:对于函数,存储过程等数据复制可能带来未知错误.
- Mixed 实际上就是前两种模式的结合。 在Mixed模式下，MySQL会根据执行的每一条具体的SQL语句来区分对待记录的日志形式， 也就是在Statement和Row之间选择一种符合当前语义的方式.

BinLog配置

```
log-bin={logName}
binlog_format="MIXED"  | "ROW"  | "STATEMENT"
```

基于Binlog的主从复制配置总结

Master操作:

1. 接入mysql并创建主从复制的用户 `create user m2ssync identified by 'Qq123!@#';`
2. 给新建的用户赋权 `GRANT REPLICATION SLAVE ON *.* TO 'm2ssync'@'%' IDENTIFIED BY 'Qq123!@#';`
3. 指定服务ID，开启binlog日志记录，在my.cnf中加入

```
server-id=137                #IP
log-bin=dbstore_binlog     #binlog 日志文件名
binlog-do-db=db_store      #记录日志的数据库
```

4. 通过SHOW MASTER STATUS;查看Master db状态.

slave操作:

1. 指定服务器ID，指定同步的binlog存储位置，在my.cnf中加入
2. 接入slave的mysql服务，并配置

```
change master to master_host='192.168.8.137',
master_port=3306,master_user='m2ssync',master_password='Qq123!@#',master_log_fil
e='db_stoere_binlog',master_log_pos=0;

start slave;
```

3. `show slave status\G ;`查看slave服务器状态