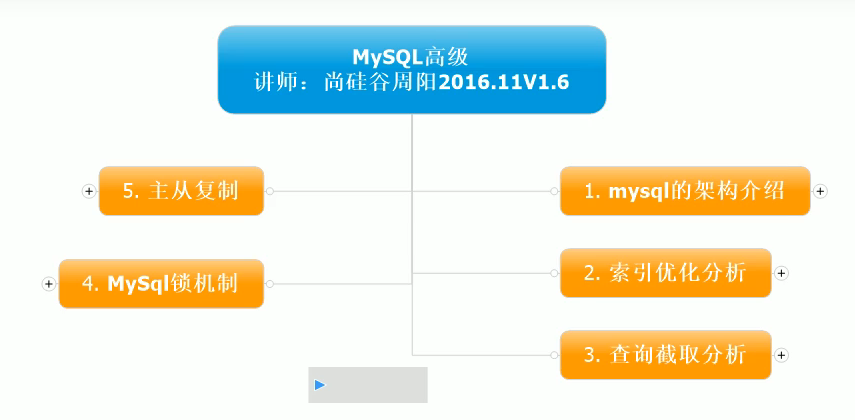
（p.179）



## （p.180）mysql的架构介绍

### 1. mysql的启停

service mysql start/stop/restart

查看版本：mysqladmin –version

### 2. 查看mysql后台是否开启

ps -ef | grep mysql

### 3. \*mysql出现乱码(p.184)

#### ①修改mysql配置文件的默认字符集

查询字符：show variables like ‘%char%’;

修改配置文件（/etc/my.cnf）：

[client]

default-character-set = utf8

[mysqld]

character-set-server = utf8

collation-server = utf8\_general\_ci

[mysql]

default-character-set = utf8

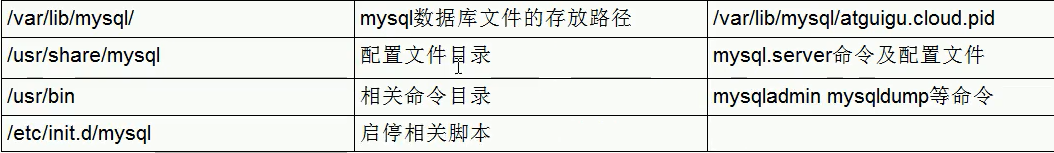
#### ②修改了编码后表查询还是出现乱码的原因：

修改配置前就建好了数据库与表，库有的默认编码，需要改变**原库编码**或者**删了重建**。

### 4. 安装位置(创建的数据库保存的目录)

windows：在安装目录的data文件夹

linux：/var/lib/mysql/



linux配置文件：/etc/my.cnf

### 5. 主要配置文件（p.185）

①二进制日志log-bin（作用：主从复制）默认关闭

②错误日志log-error（作用：记录错误信息）默认关闭

③查询日志log（记录查询的sql语句）默认关闭

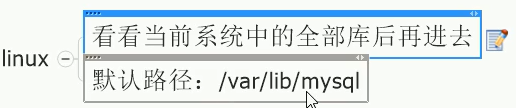
④数据文件：

windows：



**C:\ProgramData\Mysql\Mysql Server x.x\data\下（x.x为你安装的版本）**

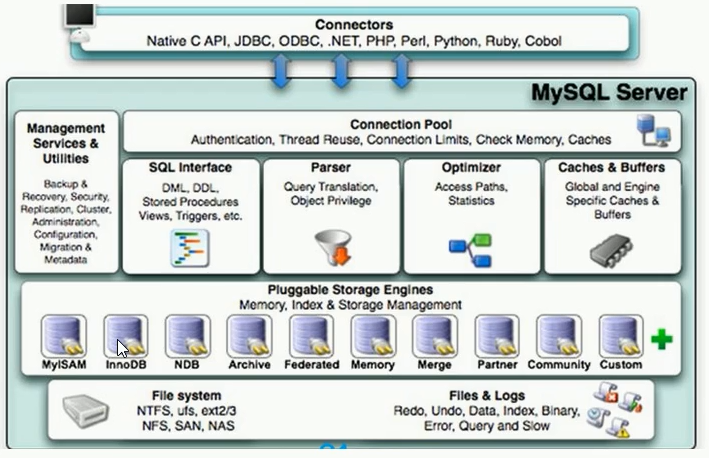
linux：

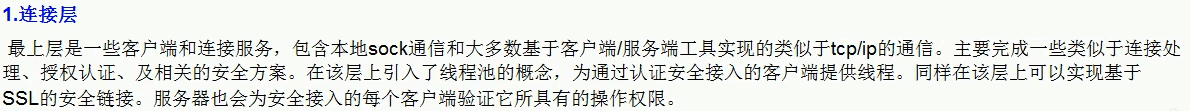


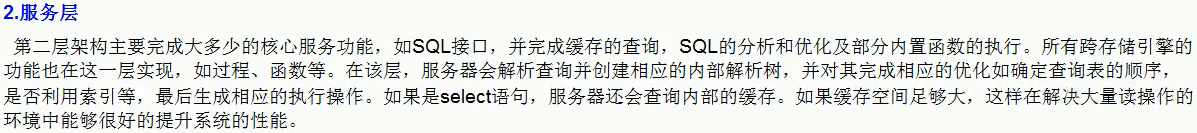
内有四种文件：

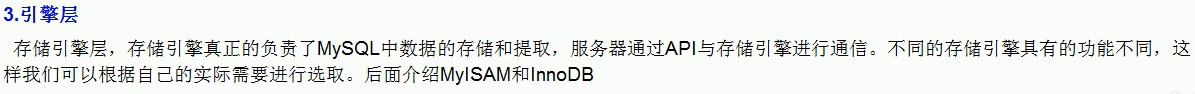
1. frm后缀：存表的框架
2. myd后缀：存数据
3. myi后缀：数据的索引

### 6. mysql的逻辑架构介绍（p.186）











### 7.存储引擎

查询支持的存储引擎：show engines;

查询当前存储引擎：show variables like '%storage\_engine%';

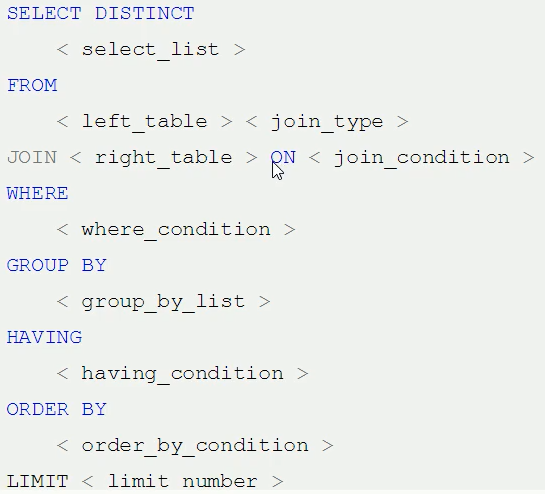


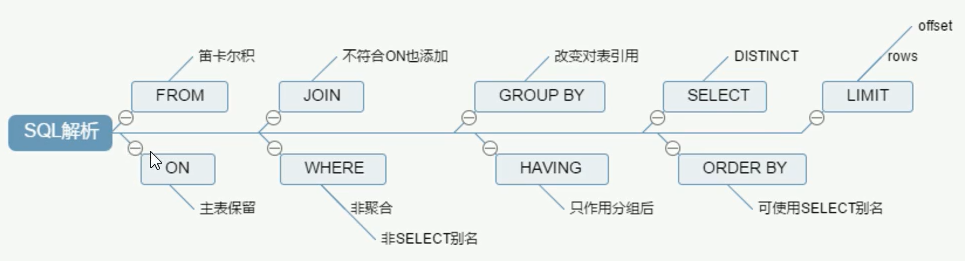
MyISAM适合读，InnoDB适合写

## （p.188 10）索引介绍

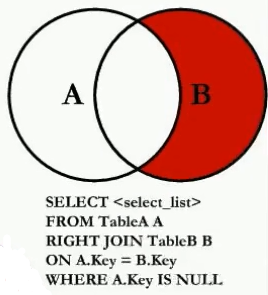
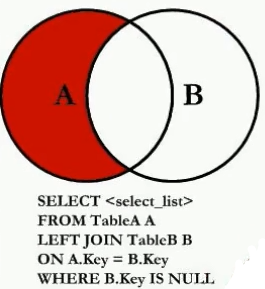
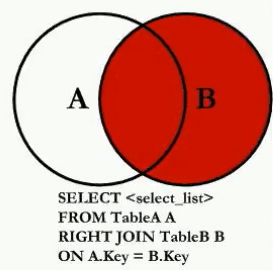
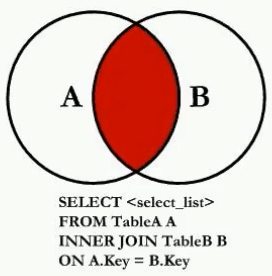
### 1. sql复习

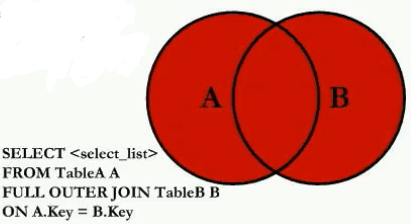
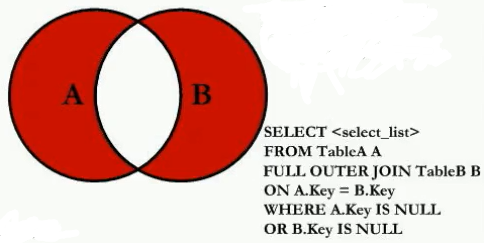
#### ①查询加载顺序



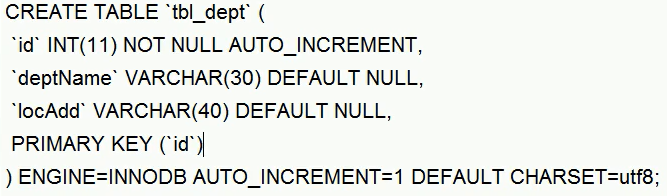


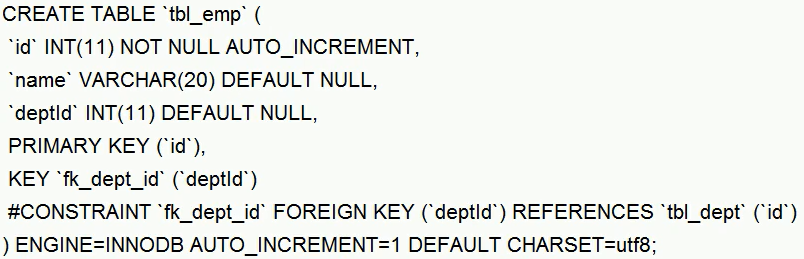
#### ②所有join图（p.191）



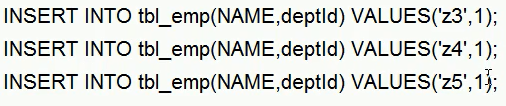
 

### 2. 建表







### 3. 性能下降，SQL慢（执行时间长，等待时间长）

①查询语句有瑕疵

②索引失效

③关联查询太多的join

④服务器调优及各个参数的设置（缓冲、线程数等）

### 4. 索引是什么（p.192 p15）

索引（index）是mysql高效获取数据的数据结构，提高查询效率。索引是排好序的快速查找数据结构。

索引会影响where后的查找与order by后面的排序（***即影响排序和查找两个功能***）。

索引未特别指明，用的是b树。聚集索引（***保存数据的物理位置，一张表只能有一个聚簇索引，一般为主键***）、次要索引、覆盖索引、复合索引、唯一索引默认使用b+树。

索引本身也很大，不可能全部存储在内存中，因此往往以索引文件的形式存储在磁盘上。

#### ①单值索引

即一个索引只包含单个列，一个表可以有多个单列索引

#### ②唯一索引

索引列的值必须唯一，但允许有空值

#### ③复合索引（建索引时，此索引优先级高于单值索引）

一个索引包含多个列

### 5. 创建索引（一般建在频繁查询的字段上）

①单值索引

create index idx\_user\_name on user(name)；

***在user表的name字段上建索引名为idx\_user\_name的索引***

②复合索引

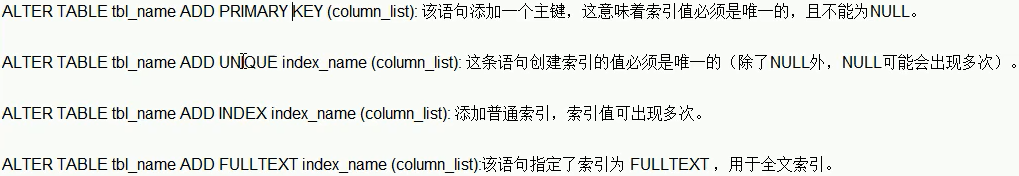
create index idx\_user\_nameEmail on user(name,email);

***在user表的name*** ***与email字段上建索引名为idx\_user\_nameEmail的联合索引***

### 6. 索引语法（p.16）

创建：create [unique] index indexName on mytable(columnname(length))

alter mytable add [unique] index [indexName] on (columnname(length))



删除：drop index [indexName] on mytable

查看：show index from tableName

### 7. mysql索引结构

①BTree索引

②Hash索引③full-text全文索引④R-Tree索引

### 8.需要建立与不需要建立索引的情况（p.18）

①需要建立索引的情况

1. 主键自动建立索引
2. 频繁作为查询条件的字段
3. 外键字段
4. 查询中需要排序的字段
5. 查询中统计或分组字段

②不需要建立索引的情况

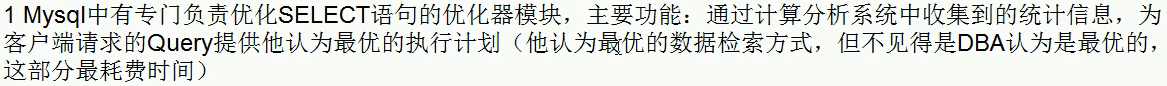
1. 频繁增删改的字段：

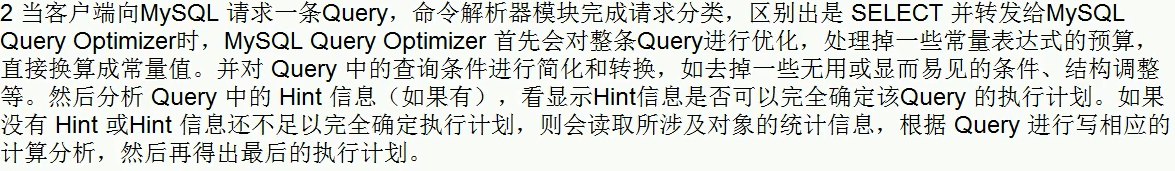
因为虽然提高了查询的速度，但同时会降低更新表的速度，更新表的时候，mysql不仅要保存数据，还要重建索引树

1. where里用不到的字段
2. 表记录太少
3. 数据中重复数据多（例如性别）

## （p.198 20）索引优化

### 1. Mysql Query Optimizer（查询优化器）





### 2. mysql常见瓶颈

①CPU在饱和时一般发生在数据装入内存或从磁盘上读取数据时

②I/O瓶颈再装入数据远大于内存容量的时候

③硬件出现瓶颈：可以通过配置文件等方式调整

### 3. explain（p.199）

使用explain关键字可以模拟优化器执行sql查询语句，从而指导mysql是如何处理你的sql并分析查询语句或是表结构的性能瓶颈

#### ①使用方式：

在查询语句前加explain

#### ②作用：

1. 表的读取顺序：id
2. 数据读取操作的操作类型
3. 哪些索引能被使用
4. 哪些索引实际被使用
5. 表之间的引用
6. 每张表有多少行被优化器查询

#### ③查询结果：

**id：读取表的顺序**

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

（1）id相同执行顺序从上到下

（2）id不同，如果是子查询，则执行顺序由大至小

**select\_type：操作类型**

1. simple：简单查询，查询中不包含子查询或union
2. primary：若查询中包含任何复杂的子查询，则最外层的为primary
3. subquery：在select或where列表中包含了子查询（无from）
4. derived：from中包含的子查询被标记为derived（**衍生**），相当于对某张表查询时的临时表
5. union：若第二个select出现在union之后，则被标记为union；若union包含在from子句的子查询中，外层select被标记为derived
6. union result：从union表获取结果的result

**table：表名**

若为from后的子查询，为 derived+一个数字，这个数字代表**衍生**表查询的id

**partitions：**

**type：访问类型（常用的以下为由好到差，并不完全，一般保证优化到range级别，最好能到ref）**

1. ***system***：表只有一行，所以查询结果只有一行（等于系统表），这是const类型的特例，平时不会出现
2. ***const***：表示通过索引一次就找到了，const用于比较primary key或者unique索引。因为只匹配一行数据，所以很快。如将逐渐置于where列表中，mysql就能将该查询转换成一个常量。
3. ***eq\_ref***：唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描，**两个表的连接条件字段均为唯一键或主键**（用于多表联结）
4. ***ref***：非唯一性索引扫描，返回匹配某个单独值的所有行。本质也是一种索引访问，它返回所有匹配某个单独值的行，然而它可能会找到多个符合条件的行，**某个非主键的索引作为条件**，所以属于查找与扫描的混合体。（组合索引第一个列中的查询）
5. ***range***：只检索给定范围的行，使用一个索引来选择行。key列显示使用了那个索引。一般在where语句中出现了between，<>，in 等的查询。这种扫描比全表扫描好。
6. ***index***：索引全扫描，一般发生在覆盖索引的时候，也就是对有索引列发生一次全扫描；或者是对索引进行排序，如：select \* from table1 order by index1。
7. ***all***：全表扫描

**possible\_keys：可能运用到的索引**

查询涉及到字段上若存在索引，则该索引被列出，但不一定被实际使用

**key：实际使用到的索引**

如果为null，则没有使用索引。查询中若使用覆盖索引，则该索引仅出现在key列表中。

possible\_keys没有，但key有的情况是：select col1,col2 from table1，而（col1，col2）正好是一个组合索引，而且在select后排列的顺序相同

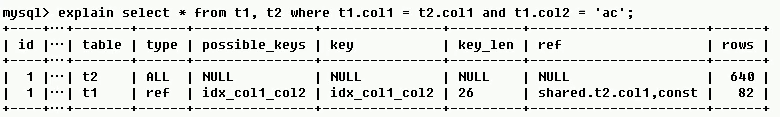
**key\_len：**

表示索引中用到的字节数，可通过该列就按查询中使用的索引的长度。在不损失精确度的情况下，长度越短越好

key\_len显示的值为索引字段的最大可能长度，并非实际使用长度，key\_len是根据表定义计算而得，不是通过表内检索得出

**ref：表示性能如何**

涉及到了那些列，如下图所示：先加载了t2表，所以为null，再加载了t1表，其中涉及到了索引哪一列被使用，这些列或常量被用来查询哪些值



**rows：**

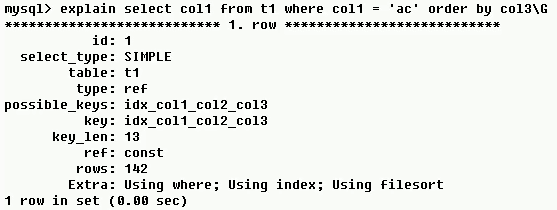
根据表统计信息及索引选用情况，大致估算出找到所需的记录所需要读取的行数

**filtered**

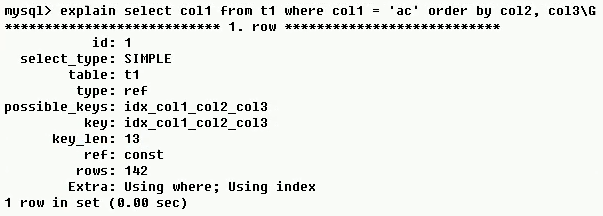
**extra：**

包含不适合在其他列中显示但十分重要的额外信息

1. **using filesort:** mysql未使用索引的顺序进行排序。 mysql中**无法利用索引完成的排序**操作称为“文件排序”，一种不好的现象。

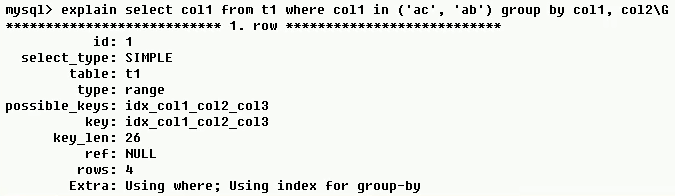


如果组合索引中跳索引了（中间某个索引列没用到），那么会产生此现象。

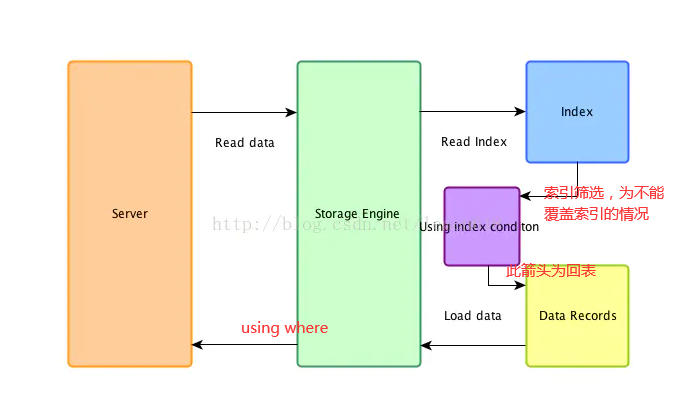


1. **using temporary**:使用了临时表保存中间结果，mysql在查询结果排序时使用临时表。常见于order by和分组查询group by（**group by后列的顺序尽量和索引列的顺序相吻合**）





1. **using index：**表示相应的查询操作使用了覆盖索引，且不用回表重新访问，而**null**表示需要回表。
2. **using where：**服务层需要对存储引擎层返回的数据进行筛选。
3. **using index condition：**仅适用于二级索引，一般发生在查询字段无法被二级索引覆盖的场景***（即组合索引（a，b，c）查询条件只给了a，c）***，该场景下往往***需要回表***。此为mysql5.6后出现的ICP（index condition pushdown）机制，此机制使用了index filter(见博客https://www.jianshu.com/p/c6483ded042d)
4. **using where;using index：**与using index condition类似，只是最后不会回表



### 4. explain单表案例（p.31）

#### ①建表：

CREATE TABLE IF NOT EXISTS `article` (

`id` INT(10) UNSIGNED NOT NULL PRIMARY KEY AUTO\_INCREMENT,

`author\_id` INT(10) UNSIGNED NOT NULL,

`category\_id` INT(10) UNSIGNED NOT NULL,

`views` INT(10) UNSIGNED NOT NULL,

`comments` INT(10) UNSIGNED NOT NULL,

`title` VARBINARY(255) NOT NULL,

`content` TEXT NOT NULL

);

INSERT INTO `article`(`author\_id`, `category\_id`, `views`, `comments`, `title`, `content`) VALUES

(1, 1, 1, 1, '1', '1'),

(2, 2, 2, 2, '2', '2'),

(1, 1, 3, 3, '3', '3');

#### ②查询 category\_id 为1 且comments 大于 1 的情况下,views 最多的 article\_id。

EXPLAIN SELECT id,author\_id

FROM article

WHERE category\_id=1 AND comments>1

ORDER BY views DESC

LIMIT 1;

需要建联合索引 (category\_id,views,comments)，而非(category\_id,comments,views);

tips : 因为当出现某列为一个范围的时候，会使**组合索引**此列**后面的列**失索引效，所以会出现using filesort

### 5. join语句优化

#### 建表：

CREATE TABLE IF NOT EXISTS `class` (

`id` INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,

`card` INT(10) UNSIGNED NOT NULL,

PRIMARY KEY (`id`)

);

CREATE TABLE IF NOT EXISTS `book` (

`bookid` INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,

`card` INT(10) UNSIGNED NOT NULL,

PRIMARY KEY (`bookid`)

);

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO class(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

INSERT INTO book(card) VALUES(FLOOR(1 + (RAND() \* 20)));

#### 内连接

关联查询时必须要有一个表为驱动表（即全查询表），mysql会自动将小表作为驱动表

#### 左、右外连接（p.32）

EXPLAIN SELECT \* FROM class LEFT JOIN book ON class.card = book.card;

结论：左、右外连接用的多的情况下，附表的连接字段加索引（因为主表的每个字段都是要显示的，所以一定是all。若使用主表，且主表的card为索引时为index，说明该主表就只有两列数据）

#### 建表（三表优化）

CREATE TABLE IF NOT EXISTS phone(

phoneid INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,

card INT(10) UNSIGNED NOT NULL,

PRIMARY KEY(phoneid)

)ENGINE=INNODB;

INSERT INTO phone(card) VALUES(FLOOR(1+RAND()\*20));

#### 三表左连接

EXPLAIN SELECT \*

FROM class

LEFT JOIN book ON class.card=book.card

LEFT JOIN phone ON book.card=phone.card

索引建在非主表的连接条件列上

#### join语句的优化

①尽可能减少join中的nestedLoop的循环总次数；

②“永远用小结果集驱动大的结果集”

③优先优化内存循环

④保证join语句中被驱动表上join条件字段已经被索引

⑤当无法保证被驱动表的join条件字段被索引且内存资源充足的前提下，不要太吝啬JoinBuffer的设置

### 6. 索引优化

#### 建表

CREATE TABLE staffs (

id INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR (24) NULL DEFAULT '' COMMENT '姓名',

age INT NOT NULL DEFAULT 0 COMMENT '年龄',

pos VARCHAR (20) NOT NULL DEFAULT '' COMMENT '职位',

add\_time TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP COMMENT '入职时间'

) CHARSET utf8 COMMENT '员工记录表' ;

INSERT INTO staffs(NAME,age,pos,add\_time) VALUES('z3',22,'manager',NOW());

INSERT INTO staffs(NAME,age,pos,add\_time) VALUES('July',23,'dev',NOW());

INSERT INTO staffs(NAME,age,pos,add\_time) VALUES('2000',23,'dev',NOW());

INSERT INTO staffs(NAME,age,pos,add\_time) VALUES(null,23,'dev',NOW());

ALTER TABLE staffs ADD INDEX idx\_staffs\_nameAgePos(name, age, pos);

#### ①原因1：全局匹配我最爱（解决索引失效p.34）

EXPLAIN SELECT \* FROM staffs WHERE NAME = 'July';

EXPLAIN SELECT \* FROM staffs WHERE NAME = 'July' AND age=25;

EXPLAIN SELECT \* FROM staffs WHERE NAME = 'July' AND age=25 AND pos='dev';

***以上三种情况由于查询字段的顺序是与建立索引的顺序一致，所以都用上了索引（但索引中的key\_len有区别）***

EXPLAIN SELECT \* FROM staffs WHERE age=25 AND pos='dev';

EXPLAIN SELECT \* FROM staffs WHERE pos='dev';

***以上两种情况就没用上索引，因为各缺失了前几列***

EXPLAIN SELECT \* FROM staffs WHERE NAME = 'July' AND pos='dev';

***此情况虽用到了索引，但是并没有对pos列使用加速查询，相当于只用了一半，在explain中的type和ref有体现***

以火车为例，第二节断了，第一节可以走而第三节不能走

#### ②最佳左前缀法则

索引是排好序的数据结构，查询从索引的最左列开始且不跳过索引中的列

带头大哥不能倒，中间兄弟不能断

#### ③不在索引列上做任何的操作（p.35）

不在索引列上做任何的操作（计算、函数（自动或手动）类型转换），会导致索引失效

EXPLAIN SELECT \* FROM staffs WHERE left(NAME,4) = 'July';

由于使用函数，以上语句会使索引idx\_staffs\_nameAgePos失效

#### ④存储引擎不能使用索引中范围条件右边的列

EXPLAIN SELECT \* FROM staffs WHERE NAME = 'July' AND age>25 AND pos='dev';

此处由于age是一个范围，所以导致idx\_staffs\_nameAgePos右边的列索引失效

#### ⑤尽量使用覆盖索引

尽量使用覆盖索引（只访问索引的查询（查询列和查询列一致）），减少使用select\*

#### ⑥mysql使用不等于（!=或者<>）时无法使用索引

#### ⑦isnull/is not null也无法使用索引

#### ⑧like以通配符开头(‘%abc’)会索引失效（p.40）

解决方法：使用覆盖索引解决（以下为name为索引字段），不回表即可

EXPLAIN SELECT NAME FROM staffs WHERE NAME LIKE '%a'

但是如果需要的字段必须回表，例如select\*，那就依然会失效

#### ⑨字符串不加单引号会使索引失效（p.41）

虽然不会报错但是会进行隐形的类型转换，转③

#### ⑩少用or

or可能会导致索引失效

### 7. 索引面试题分析（p.44）

#### 建表：

create table test03(

id int primary key not null auto\_increment,

c1 char(10),

c2 char(10),

c3 char(10),

c4 char(10),

c5 char(10)

);

insert into test03(c1,c2,c3,c4,c5) values('a1','a2','a3','a4','a5');

insert into test03(c1,c2,c3,c4,c5) values('b1','b2','b3','b4','b5');

insert into test03(c1,c2,c3,c4,c5) values('c1','c2','c3','c4','c5');

insert into test03(c1,c2,c3,c4,c5) values('d1','d2','d3','d4','d5');

insert into test03(c1,c2,c3,c4,c5) values('e1','e2','e3','e4','e5');

#### 案例1

EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c2=''AND c3=''AND c4='';

EXPLAIN SELECT \* FROM test03 WHERE c4=''AND c3=''AND c2=''AND c1='a1';



以上两种结果都一样，where后面字段的排列顺序对索引无影响，因为优化器

EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c2=''AND c3>''AND c4=''



以上仅用到三个索引

EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c2=''AND c4='' ORDER BY c3



c1，c2用到了，c3、c4用来index filter(此用处来在索引中进行过滤与排序)

***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c2='' ORDER BY c3***



对查找来说用到了c1、c2，对排序来说其实c3也用到了（index filter）

***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c2='' ORDER BY c4***



由于中间断了c3，所以在索引中对c1、c2查询的索引来说c4是无序的，所以出现了using filesort

***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c5='' ORDER BY c2,c3***



查找只用了c1，c2、c3用来index filter，c5对搜索引擎返回数据using where

***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c5='' ORDER BY c3,c2***



***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c4='' GROUP BY c2,c3***



***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c4='' ORDER BY c2,c3***



***EXPLAIN SELECT \* FROM test03 WHERE c1='a1' AND c4='' GROUP BY c3,c2***



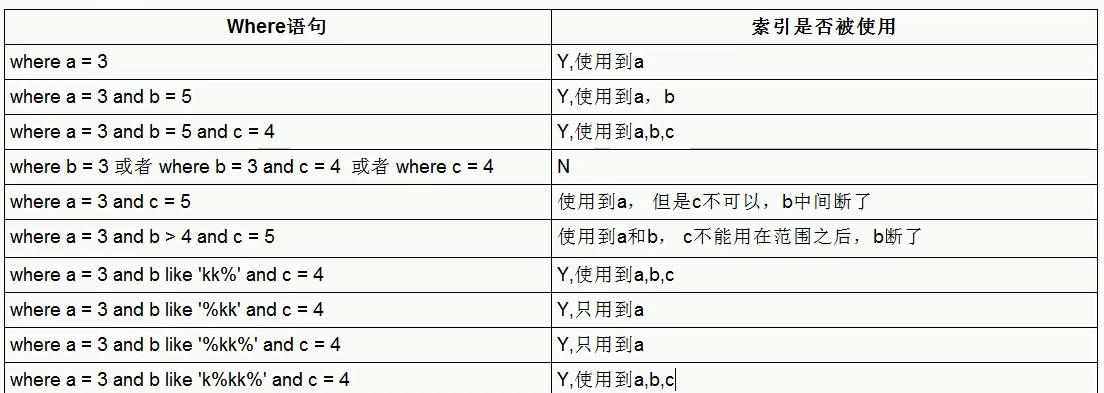
分组之前必排序

***EXPLAIN SELECT \* FROM test03 WHERE c1=''AND c2 LIKE 'k%'AND c3=''***

***EXPLAIN SELECT \* FROM test03 WHERE c1=''AND c2 LIKE 'k%kk%'AND c3=''***



like ‘k%’不会使索引后面的字段失效，而范围会



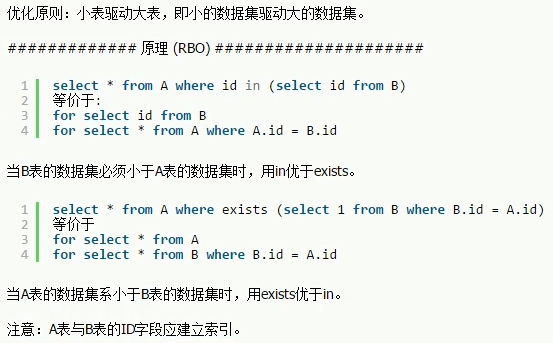
## （p.224 46）查询截取分析

分析：

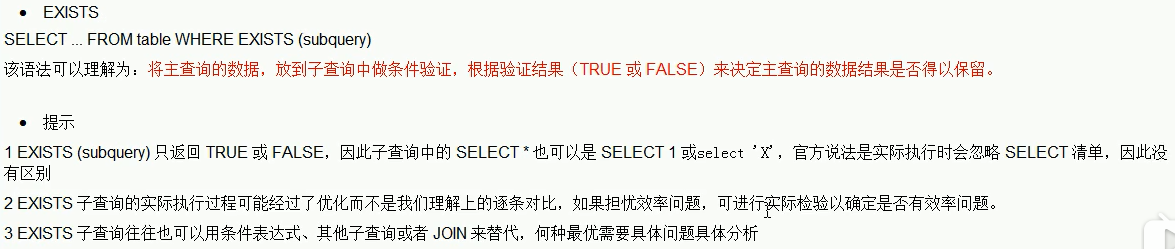
1. 观察一天，看看生产慢sql的情况
2. 开启慢查询日志，设置阈值，比如超过5s的就是慢sql，并抓取
3. explain+慢SQL分析（此处可以解决80%的问题）
4. show profile查询SQL在mysql服务器里的执行细节和生命周期情况
5. 运维经理 or DBA，进行SQL数据库服务器的参数调优

### 查询优化

#### ①永远小表驱动大表（in和exists（p.46））



因为是将子查询先查出来，然后当做驱动表。in中内表为驱动表，exists中外表为驱动表。



SELECT \*

FROM tb1\_emp e

WHERE e.deptId IN(SELECT id

FROM tb1\_dept d);

SELECT \*

FROM tb1\_emp e

WHERE EXISTS(SELECT 1

FROM tb1\_dept d WHERE e.deptId=d.id);

以上两种写法结果集相同

#### ②order by关键字

尽量使用index方式排序，避免使用filesort方式排序

***建表：***

CREATE TABLE tblA(

id INT PRIMARY KEY NOT NULL AUTO\_INCREMENT,

age INT,

birth TIMESTAMP NOT NULL,

NAME VARCHAR(200)

);

INSERT INTO tblA(age,birth,NAME) VALUES(22,NOW(),'abc');

INSERT INTO tblA(age,birth,NAME) VALUES(23,NOW(),'bcd');

INSERT INTO tblA(age,birth,NAME) VALUES(24,NOW(),'def');

CREATE **INDEX** idx\_A\_ageBirth ON **tblA(age,birth)**;

EXPLAIN SELECT \* FROM tblA WHERE age>20 ORDER BY age;用不到索引

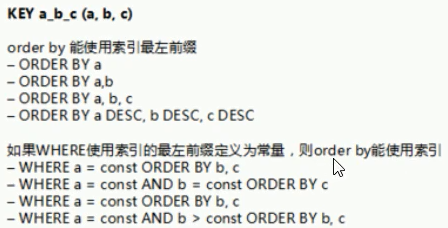
EXPLAIN SELECT age FROM tblA WHERE age>20 ORDER BY age;可以用索引

***order by的建议：***

①在SELECT中查询了索引建以外的列，那么ORDER BY就不会使用索引了（则排序必须要***覆盖索引***或者***强制使用索引***，不然会用不上索引，出现FileSort），***尽量别用select \****

②在覆盖索引中，order by语句使用索引最左前列原则，使用where字句与order by字句条件列满足索引最左前列。

③若必须使用fileort的话，即排序列真不在索引列上，filesort有两种算法：双路排序、单路排序，单路算法可能会失效，从而导致多次io性能甚至低于双路（**但是通过调整参数sort\_buffer、max\_length\_for\_sort\_data值的大小可以避免此现象**）。



#### ③group by

group by实际是线排序后分组，遵照索引的最佳左前缀

若无法使用索引(运用到filesort时)，仍然需要增大max\_length\_for\_sort\_data和sort\_buffer\_size

where能解决的，就不要用having

### 慢查询日志（p.49）

#### ①是什么

此为mysql提供的一种日志记录，用来记录在mysql响应时间超过阈值的语句，具体运行时间是指大于long\_query\_time值的sql，则被记录到慢查询日志

1）默认mysql未开启慢查询日志，需要我们手动开启

2）若不调优，不建议打开，打开会对性能有一些影响

#### ②开启方式

***开启慢查询日志与查询慢查询日志保存地址***

set global slow\_query\_log=1;（此方式重启mysql后无效）

***永久生效的方式需要修改配置文件my.cnf***

slow\_query\_log =1

slow\_query\_log\_file=/var/lib/mysql/atguigu-slow.log

***查询与修改慢的时间（也可以在配置文件中修改）***

show variables like '%long\_query\_time%';

修改： set global long\_query\_time=3; 后重开会话生效

***查询当前共记录下多少条慢查询sql***

show global status like '%slow\_queries%';

**配置文件版：**

slow\_query\_log=1;

slow\_query\_log\_file=/var/lib/mysql/atguigu-slow.log

long\_query\_time=3;

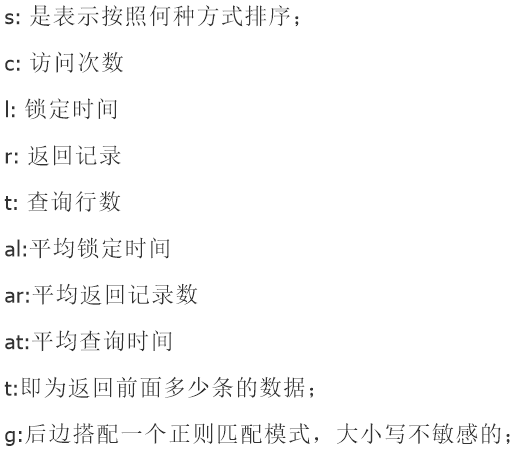
log\_output=FILE

#### ③日志分析工具mysqldumpslow

在生产环境中，提供了日志分析工具mysqldumpslow来分析sql语句

***查询mysqldumpslow的命令：（非mysql客户端中）***

mysqldumpslow –help



**1）得到返回记录集最多的10个SQL**

mysqldumpslow -s r -t 10 /var/lib/mysql/iz2zej55532kgul2qme7cwz-slow.log

**2）得到访问次数最多的10个SQL**

mysqldumpslow -s c -t 10 /var/lib/mysql/iz2zej55532kgul2qme7cwz-slow.log

**3）得到按照时间排序的前10条里面含有左连接的查询语句**

mysqldumpslow -s t -t 10 -g "left join" /var/lib/mysql/iz2zej55532kgul2qme7cwz-slow.log

**4）另外建议在使用这些命令时结合 | 和more 使用 ，否则有可能出现爆屏情况**

mysqldumpslow -s r -t 10 /var/lib/mysql/iz2zej55532kgul2qme7cwz-slow.log | more

### 插入大量数据建表

#### ①建表

# 新建库

create database bigData;

use bigData;

#1 建表dept

CREATE TABLE dept(

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

deptno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0,

dname VARCHAR(20) NOT NULL DEFAULT "",

loc VARCHAR(13) NOT NULL DEFAULT ""

) ENGINE=INNODB DEFAULT CHARSET=UTF8 ;

#2 建表emp

CREATE TABLE emp

(

id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

empno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0, /\*编号\*/

ename VARCHAR(20) NOT NULL DEFAULT "", /\*名字\*/

job VARCHAR(9) NOT NULL DEFAULT "",/\*工作\*/

mgr MEDIUMINT UNSIGNED NOT NULL DEFAULT 0,/\*上级编号\*/

hiredate DATE NOT NULL,/\*入职时间\*/

sal DECIMAL(7,2) NOT NULL,/\*薪水\*/

comm DECIMAL(7,2) NOT NULL,/\*红利\*/

deptno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0 /\*部门编号\*/

)ENGINE=INNODB DEFAULT CHARSET=UTF8 ;

#### ②创建插入海量数据可能出现的错误

创建函数，假如报错：This function has none of DETERMINISTIC......

***1）由于开启过慢查询日志，所以需要开启bin-log（二进制日志）, 我们就必须为我们的function指定一个参数。***

show variables like 'log\_bin\_trust\_function\_creators';

set global log\_bin\_trust\_function\_creators=1;

***2）配置文件修改方法：***

windows下my.ini[mysqld]加上log\_bin\_trust\_function\_creators=1

linux下 /etc/my.cnf下my.cnf[mysqld]加上log\_bin\_trust\_function\_creators=1

#### ③创建函数（随机产生字符串与部门编号）

1. ***随机产生英文***

DELIMITER $

CREATE FUNCTION rand\_string(n INT) RETURNS VARCHAR(255)

BEGIN ##方法开始

DECLARE chars\_str VARCHAR(100) DEFAULT 'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';

##声明一个 字符窜长度为 100 的变量 chars\_str ,默认值

DECLARE return\_str VARCHAR(255) DEFAULT '';

DECLARE i INT DEFAULT 0;

##循环开始

WHILE i < n DO

SET return\_str =CONCAT(return\_str,SUBSTRING(chars\_str,FLOOR(1+RAND()\*52),1));

##concat 连接函数 ，substring(a,index,length) 从index处开始截取

SET i = i + 1;

END WHILE;

RETURN return\_str;

END $

1. ***随机产生数字***

DELIMITER $

CREATE FUNCTION rand\_num( )

RETURNS INT(5)

BEGIN

DECLARE i INT DEFAULT 0;

SET i = FLOOR(100+RAND()\*10);

RETURN i;

END $

#### ④创建存储过程（插入数据的存储过程）

1. ***向emp表插入数据***

DELIMITER $$

CREATE PROCEDURE insert\_emp10000(IN START INT(10),IN max\_num INT(10))

BEGIN

DECLARE i INT DEFAULT 0;

#set autocommit =0 把autocommit设置成0 ；提高执行效率

SET autocommit = 0;

REPEAT ##重复

SET i = i + 1;

INSERT INTO emp10000 (empno, ename ,job ,mgr ,hiredate ,sal ,comm ,deptno ) VALUES ((START+i) ,rand\_string(6),'SALESMAN',0001,CURDATE(),FLOOR(1+RAND()\*20000),FLOOR(1+RAND()\*1000),rand\_num());

UNTIL i = max\_num ##直到 上面也是一个循环

END REPEAT; ##满足条件后结束循环

COMMIT; ##执行完成后一起提交

END $$

1. ***向dept表插入数据***

#执行存储过程，往dept表添加随机数据

DELIMITER $$

CREATE PROCEDURE insert\_dept(IN START INT(10),IN max\_num INT(10))

BEGIN

DECLARE i INT DEFAULT 0;

SET autocommit = 0;

REPEAT

SET i = i + 1;

INSERT INTO dept (deptno ,dname,loc ) VALUES (START +i ,rand\_string(10),rand\_string(8));

UNTIL i = max\_num

END REPEAT;

COMMIT;

END $$

#### ⑤插入数据

CALL insert\_dept(100,10);

CALL insert\_emp(100001,500000);

CALL insert\_emp10000(100001,10000);

### show Profile（p.51）

是mysql提供可以用来分析当前会话中语句执行的资源消耗情况，默认关闭，并保存最近15此的运行结果。

#### ①开启show profile

1. ***查看当前版本是否支持***

show variables like 'profiling';

***2）打开功能***

set profiling =1;

#### ②使用

可能会出现ONLY\_FULL\_GROUP\_BY的错误（sqlyog无此错误），此错误为group by后的字段必须出现在select后。

解决方式：

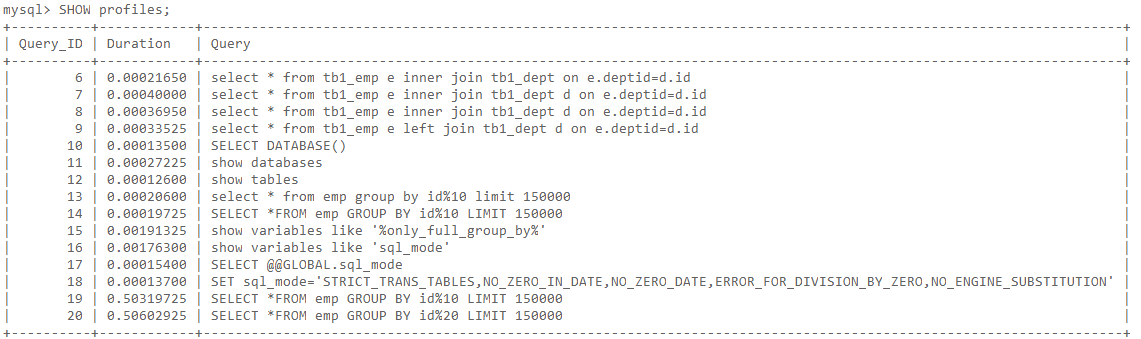
SET sql\_mode='STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

<https://blog.csdn.net/free377096858/article/details/91963406>

1. ***查看数据***

show profiles;

显示当前会话所有查询语句的记录（sqlyog可能会出现错误）



***2）诊断***

show profile cpu,block io for query n (n为上一步前面的问题SQL数字号码);

type:

| ALL --显示所有的开销信息

| BLOCK IO --显示块IO相关开销

| CONTEXT SWITCHES --上下文切换相关开销

| CPU --显示CPU相关开销信息

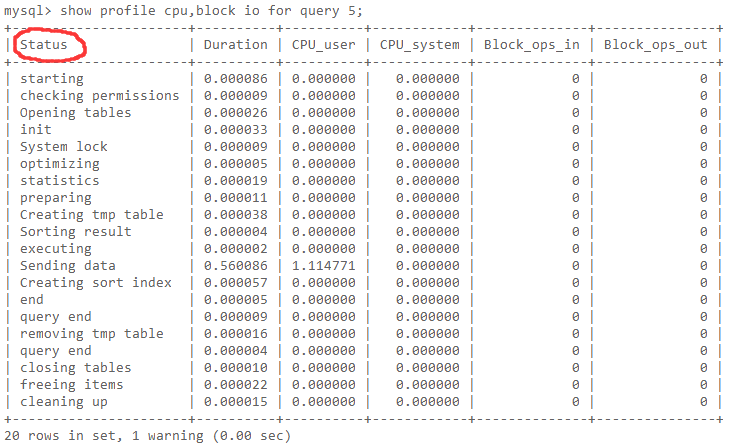
| IPC --显示发送和接收相关开销信息

| MEMORY --显示内存相关开销信息

| PAGE FAULTS --显示页面错误相关开销信息

| SOURCE --显示和Source\_function，Source\_file，Source\_line相关的开销信息

| SWAPS --显示交换次数相关开销的信息



若是在status中出现了以下四种情况则表示比较危险

**converting HEAP to MyISAM** 查询结果太大，内存都不够用了往磁盘上搬了。

**Creating tmp table**  步骤：①创建临时表②拷贝数据到临时表 ③用完再删除

**Copying to tmp table on disk** 把内存中临时表复制到磁盘

**locked**

### 全局查询日志（生产环境不能开!p.52）

记录全局所有的sql，***包括别的会话***；

#### ①开启方式

***1）配置启用***

在mysql的my.cnf中，设置如下：

#开启

general\_log=1

# 记录日志文件的路径

general\_log\_file=/path/logfile

#输出格式

log\_output=FILE

***2）会话开启***

set global general\_log=1;

set global log\_output='TABLE';（以表的形式记录，保存到mysql库的general\_log表中）

## （p.231 53）mysql数据库的锁

### 1. 锁的分类

①对数据操作的类型分（读锁、写锁）

1. 读锁（共享）：针对同一份数据，多个读操作可以同时进行而不会互相影响
2. 写锁（排他）：当前写操作没有完成前，它会阻断其他写锁和读锁

②对数据操作的粒度分（表锁，行锁）

### 2. 表锁（偏向MyISAM）p.54-55

偏向MyISAM引擎，开销小，加锁快，无死锁；锁定力度大，发生冲突的概率高，并发度低。

MyISAM在***执行select语句***时，会***自动***给涉及的所有表***添加读锁***；在执行***增删改***操作前，会给涉及的表加上***写锁***。

***以下为手动加锁的命令。***

#### ①建表

create table mylock(

id int not null primary key auto\_increment,

name varchar(20)

)engine myisam;

insert into mylock(name) values('a');

insert into mylock(name) values('b');

insert into mylock(name) values('c');

insert into mylock(name) values('d');

insert into mylock(name) values('e');

#### ②表锁命令

***1）加锁***

lock table 表名字1 read(write)，表名字2 read(write)，其它;

***2）查看哪些表被锁了***

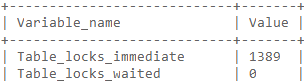
show open tables;

***3）释放表锁***

unlock tables;

***4）查看表等待次数***

show status like 'table%';



第一个是立即获取锁的次数；

第二个是表级锁定争用时发生等待的次数，此值高表示表级锁争用情况

#### ③读锁demo（read）

|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：对表mylock加read锁 | 第一步：连接时，session1已经锁定了mylock表 |
| 第二步：查询mylock（√） | 第二步：查询mylock（√） |
| 第三步：更改mylock（×）  Update、insert、delete皆不可 | 第三步：更改mylock表  会阻塞，直到session1释放锁。 |
| 第四步：查看mylock以外别的表（×）  必须将锁定的表释放，才可以查看别的表 | 第四步：增删查改unlock以外的表（√） |

#### ④写锁demo（write）

|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：对表mylock加write锁 | 第一步：连接时，session1已经锁定了mylock表 |
| 第二步：查询mylock（√） | 第二步：查询mylock  阻塞，直到session1释放锁  Tips：有时候由于mysql的***缓存机制***，可能会查到数据，但是这时候将where条件改变就会阻塞。 |
| 第三步：更改mylock（√） | 第三步：更改mylock表  会阻塞，直到session1释放锁。 |
| 第四步：查看/修改mylock以外别的表（×）  必须将锁定的表释放，才可以查看别的表 | 第四步：增删查改unlock以外的表（√） |

#### ⑤读写锁小总结

读锁会阻塞写，写锁会阻塞读和写。

MyISAM的读写调度是写优先，所以***不适合***作为***操作多***的引擎；因为写锁会使其他会话不能进行操作，大量更新会使查询阻塞，效率低。

### 3. 行锁（偏向InnoDB存储引擎）

开销大，加锁慢；会出现死锁；锁定粒度较小，发生锁冲突的概率低，并发量高

InnoDB与MyISAM最大的不同是***InnoDB支持事务***且***采用行锁***

#### ①建表

create table test\_innodb\_lock (a int(11),b varchar(16))engine=innodb;

insert into test\_innodb\_lock values(1,'b2');

insert into test\_innodb\_lock values(3,'3');

insert into test\_innodb\_lock values(4,'4000');

insert into test\_innodb\_lock values(5,'5000');

insert into test\_innodb\_lock values(6,'6000');

insert into test\_innodb\_lock values(7,'7000');

insert into test\_innodb\_lock values(8,'8000');

insert into test\_innodb\_lock values(9,'9000');

insert into test\_innodb\_lock values(1,'b1');

create index test\_innodb\_a\_ind on test\_innodb\_lock(a);

create index test\_innodb\_lock\_b\_ind on test\_innodb\_lock(b);

#### ②隔离级别

由于mysql每个会话默认的隔离级别是Repeatable Read

在innodb存储引擎下打开事务：

set autocommit=0;

#### ③行锁演示（需经过上一步）

|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：打开事务  Set autocommit=0; | 第一步：打开事务  Set autocommit=0; |
| 第二步：更改test\_innodb\_lock中a=4的数据，但不提交（√） | 第二步：session1更改test\_innodb\_lock中a=4但不提交，session2更改a=4的数据  则会阻塞，直到session1提交事务 |
| 第三步：commit; | 第三步：session1提交事务后，释放行锁，session2结束阻塞并进行修改。 |

但是如果两个session在自己的事务内修改***同一张表***的***不同行***就不会造成阻塞。

### 4. 无索引行锁升级成表锁（p.59）

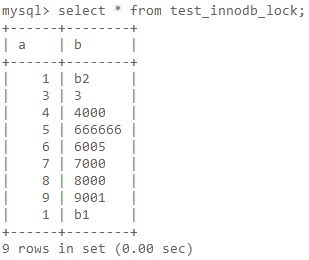
|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：打开事务  Set autocommit=0; | 第一步：打开事务  Set autocommit=0; |
| 第二步：更改test\_innodb\_lock表中a=4的数据，但是故意将varchar类型的b不加引号，且不提交（√） | 第二步：经过session1左边第二步的操作后， session2更改a=9的数据  则会阻塞，直到session1提交事务。 |
| 第三步：commit; | 第三步：session1提交事务后， session2结束阻塞并进行修改。 |

索引失效的情况下（即varchar少打引号），mysql5.7版本会将行锁变成表锁

### 5. 间隙锁的危害

若我们使用***范围条件***来***检索并修改***数据时，并***请求***共享或排它锁时，innodb会给符合条件的已有数据记录的索引项***加锁***；对于键值***在条件范围内但不存在***的记录，称为“间隙（gap）”。

Innodb也会对“间隙”加锁，这种锁机制称为***间隙锁***



上图所示，a列1~6中缺少2，2即为间隙。

|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：打开事务  Set autocommit=0; | 第一步：打开事务  Set autocommit=0; |
| 第二步：更改test\_innodb\_lock表中a的范围1~6的数据，且不提交（√） | 第二步：经过***session1左边第二步***的操作后， session2更改a=2的数据  则会阻塞，直到session1提交事务。 |
| 第三步：commit; | 第三步：session1提交事务后， session2结束阻塞并进行修改。 |

由于query执行过程中若通过范围查询，就会锁定整个范围内的所有索引键值，即使这个键值并不存在。别的会话若是想往其中出入数据时，就会阻塞，在某些场景下可能会对性能造成很大的影响。

### 如何锁定一行与查询当前行锁的情况

select …… for update（排它锁）

select …… lock for share mode（共享锁）

#### ①排它锁，写锁

一个事务对一行获取**排它锁**后，其他事务不能对此行获得任何锁（会阻塞），但是可以查询。

若是某行已被某事务获取了排它锁，其他事务修改该行时会阻塞。

|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：打开事务Set autocommit=0; | 第一步：打开事务Set autocommit=0; |
| 第二步：开始并锁定需要的行 | 第二步：***session1左边第二步***的操作后， session2更改a=8的数据  则会阻塞，直到session1提交事务。 |
| 第三步：对锁定行进行自己想要的操作 | 第三步： 阻塞 |
| 第四步：commit; 提交并释放锁 | 第四步：完成修改 |

#### ②共享锁，读锁

一个事务对一行获得共享锁后，其他事务可以对该行获得共享锁。

1）若***某行***只被***一个事务***获得了共享锁，那么该事务***可以对此行进行修改。***

2）若是***多个事务***同时获得了***某一行***的的***共享锁***，若***其中一个事务***欲***修改***该行，则会***阻塞***，若其中***两个事务欲同时***修改该行，则***后获得共享锁***的事务报***死锁并释放锁***。

|  |  |
| --- | --- |
| Session1 | Session2 |
| 第一步：打开事务Set autocommit=0; | 第一步：打开事务Set autocommit=0; |
| 第二步：开始并锁定a=2行 | 第二步：开始并锁定a=2行 |
| 第三步：更新a=2行（阻塞） | 第三步：无操作 |
| 第四步： session2第四步先行，由于右边死锁并释放锁，所以修改成功 | 第四步：更新a=2行，死锁并释放锁 |

本案例中只有两个会话获得了a=2行的锁，所以当session2释放锁后，a=2行的共享锁仅被session1获得，所以session1的第四步可以完成对数据的修改。

#### ③查询当前数据库行锁的阻塞情况

mysql> show status like 'innodb\_row\_lock%';

+-------------------------------+--------+

| Variable\_name | Value |

+-------------------------------+--------+

| Innodb\_row\_lock\_current\_waits | 0 |

| Innodb\_row\_lock\_time | 512848 |

| Innodb\_row\_lock\_time\_avg | 25642 |

| Innodb\_row\_lock\_time\_max | 51010 |

| Innodb\_row\_lock\_waits | 20 |

+-------------------------------+--------+

5 rows in set (0.00 sec)

Innodb\_row\_lock\_current\_waits：当前正在等待锁定的数量；

Innodb\_row\_lock\_time：从系统启动到现在锁定总时间长度；

Innodb\_row\_lock\_time\_avg：每次等待所花平均时间；

Innodb\_row\_lock\_time\_max：从系统启动到现在等待最常的一次所花的时间；

Innodb\_row\_lock\_waits：系统启动后到现在总共等待的次数；

#### ④行锁优化分析

1. 尽可能让所有数据检索都通过索引，避免无索引行锁变表锁
2. 合理设计检索条件，避免间隙锁
3. 尽量控制每个事务的大小，减少锁定资源量和时间长度
4. 锁住某行后，尽量别去调用别的行或表，尽快处理被锁住的行然后释放掉锁
5. 涉及相同表的事务，对于调用表的顺序尽量保持一致
6. 在业务环境允许的情况下，尽可能降低事务隔离级别

## （p.241 63）mysql主从复制

主从复制是从机向主机请求数据，而非主机向从机发送数据，所以从机可以开启防火墙，关闭端口，但是主机必须暴露端口。

### 1. 主从复制的原理

①master将改变记录到二进制日志中（binary log）。这些记录称为二进制日志事件，binary log event;

②slave将master的binary log events拷贝到它的中继日志中（relay log）；

③slave重做中继日志中的事件，将改变应用到自己的数据库中。mysql复制是异步的且串行化的

### 2. 复制的基本原则

①每个slave只能有一个master

②每个slave只能有一个唯一的服务器ID

③每个master可以有多个slave

### 3. 一主一从常见配置

#### ①版本尽量统一

#### ②两个能ping通

#### ③主机修改配置文件（配置在[mysqld]结点下面）

1）主服务器唯一id（必须）：server-id=1

2）启用二进制日志文件（必须）：log-bin=自己本地的路径/data/mysqlbin

例如：log-bin=D:/devSoft/MySQLServer5.5/data/mysqlbin

3）启用错误日志（可选）：log-err=自己本地的路径/data/mysqlerr

例如：log-err=D:/devSoft/MySQLServer5.5/data/mysqlerr

4）根目录（可选）：basedir="C:/MySQL/MySQL Server 5.5/"

5）临时目录（可选）：tmpdir="C:/MySQL/MySQL Server 5.5/"

6）数据目录（可选）：datadir="C:/ProgramData/MySQL/MySQL Server 5.5/Data/"

7）主机读写皆可（可选）：read-only=0

8）设置不需要复制的数据库（可选）：binlog-ignore-db=mysql

9）设置需要复制的数据库（可选）：binlog-do-db=需要复制的主数据库名字

#### ④配置从机（配置在[mysqld]结点下面）

1）配置二进制日志（可选）：log-bin=mysql-bin

2）配置server-id（必须）：server-id=2

#### ⑤主从机上的mysql皆重启

#### ⑥主从机皆关闭防火墙

#### ⑦主机上建立账户并授权slave

1. 授权从机登陆的账号密码（下是以账号为zhangsan，密码为123456登陆）

GRANT REPLICATION SLAVE ON \*.\* TO 'zhangsan'@'从机器数据库IP' IDENTIFIED BY '123456';

1. 刷新权限

flush privileges;

1. 查询主机状态（找出从机需要复制的文件与位置）

SHOW MASTER STATUS;



#### ⑧从机授权（最后一行从上一步中获取）

1）配置需要配置的主机

CHANGE MASTER TO MASTER\_HOST='106.14.177.207',

MASTER\_USER='root',

MASTER\_PASSWORD='root',

MASTER\_LOG\_FILE='mysqlbin.000001',MASTER\_LOG\_POS=893;

2）从机开启主从复制

start slave;

#### ⑨从机查看是否配置成功

show slave status\G;

这两处为yes表示配置成功

#### ⑩从机停止主从复制

stop slave;

若从机想重启复制，则需要回第七步重新获得主机的文件号与行号，再start，否则不能复制。