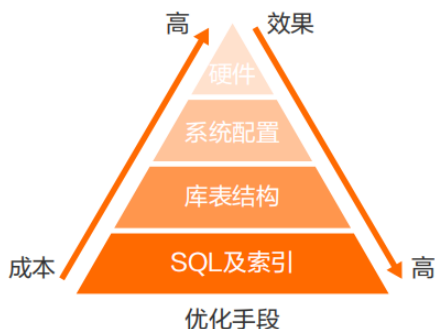


注意：本节课建议看笔记最下方的有道云网页链接笔记，pdf格式显示sql结果会有点乱，复制sql请不要在pdf里复制，会有问题，请直接在有道云网页链接笔记里复制

## Mysql全局优化总结



从上图可以看出SQL及索引的优化效果是最好的，而且成本最低，所以工作中我们要在这块花更多时间。

### 补充一点配置文件my.ini或my.cnf的全局参数：

假设服务器配置为：

- CPU：32核
- 内存：64G
- DISK：2T SSD

下面参数都是服务端参数，默认在配置文件的 [mysqld] 标签下

```
1 max_connections=3000
```

连接的创建和销毁都需要系统资源，比如内存、文件句柄，业务说的支持多少并发，指的是每秒请求数，也就是QPS。

一个连接最少占用内存是256K，最大是64M，如果一个连接请求数据超过64MB（比如排序），就会申请临时空间，放到硬盘上。

如果3000个用户同时连上mysql，最小需要内存 $3000 \times 256KB = 750M$ ，最大需要内存 $3000 \times 64MB = 192G$ 。

如果innodb\_buffer\_pool\_size是40GB，给操作系统分配4G，给连接使用的最大内存不到20G，如果连接过多，使用的内存超过20G，将会产生磁盘SWAP，此时将会影响性能。连接数过高，不一定带来吞吐量的提高，而且可能占用更多的系统资源。

```
1 max_user_connections=2980
```

允许用户连接的最大数量，剩余连接数用作DBA管理。

```
1 back_log=300
```

MySQL能够暂存的连接数量。如果MySQL的连接数达到max\_connections时，新的请求将会被存在堆栈中，等待某一连接释放资源，该堆栈数量即back\_log，如果等待连接的数量超过back\_log，将被拒绝。

```
1 wait_timeout=300
```

指的是app应用通过jdbc连接mysql进行操作完毕后，空闲300秒后断开，默认是28800，单位秒，即8个小时。

```
1 interactive_timeout=300
```

指的是mysql client连接mysql进行操作完毕后，空闲300秒后断开，默认是28800，单位秒，即8个小时。

```
1 innodb_thread_concurrency=64
```

此参数用来设置innodb线程的并发数，默认值为0表示不被限制，若要设置则与服务器的CPU核心数相同或是CPU的核心数的2倍，如果超过配置并发数，则需要排队，这个值不宜太大，不然可能会导致线程之间锁争用严重，影响性能。

```
1 innodb_buffer_pool_size=40G
```

innodb存储引擎buffer pool缓存大小，一般为物理内存的60%-70%。



```

25
26
27 # ===MySQL 8.0演示===
28 mysql> create table t1(c1 int,c2 int,index idx_c1_c2(c1,c2 desc));
29 Query OK, 0 rows affected (0.02 sec)
30
31 mysql> insert into t1 (c1,c2) values(1, 10),(2,50),(3,50),(4,100),(5,80);
32 Query OK, 5 rows affected (0.02 sec)
33
34 mysql> show create table t1\G
35 ***** 1. row *****
36 Table: t1
37 Create Table: CREATE TABLE `t1` (
38   `c1` int DEFAULT NULL,
39   `c2` int DEFAULT NULL,
40   KEY `idx_c1_c2` (`c1`,`c2` DESC) --注意这里的区别, 降序索引生效了
41 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
42 1 row in set (0.00 sec)
43
44 mysql> explain select * from t1 order by c1,c2 desc; --Extra字段里没有filesort文件排序, 充分利用了降序索引
45 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
46 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
47 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
48 | 1 | SIMPLE | t1 | NULL | index | NULL | idx_c1_c2 | 10 | NULL | 1 | 100.00 | Using index |
49 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
50 1 row in set, 1 warning (0.00 sec)
51
52 mysql> explain select * from t1 order by c1 desc,c2; --Extra字段里有Backward index scan, 意思是反向扫描索引;
53 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
54 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
55 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
56 | 1 | SIMPLE | t1 | NULL | index | NULL | idx_c1_c2 | 10 | NULL | 1 | 100.00 | Backward index scan; Using index |
57 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
58 1 row in set, 1 warning (0.00 sec)
59
60 mysql> explain select * from t1 order by c1 desc,c2 desc; --Extra字段里有filesort文件排序, 排序必须按照每个字段定义的排序或按相反顺序才能充分利用索引
61 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
62 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
63 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
64 | 1 | SIMPLE | t1 | NULL | index | NULL | idx_c1_c2 | 10 | NULL | 1 | 100.00 | Using index; Using filesort |
65 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
66 1 row in set, 1 warning (0.00 sec)
67
68 mysql> explain select * from t1 order by c1,c2; --Extra字段里有filesort文件排序, 排序必须按照每个字段定义的排序或按相反顺序才能充分利用索引

```

```

69 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
70 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
71 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
72 | 1 | SIMPLE | t1 | NULL | index | NULL | idx_c1_c2 | 10 | NULL | 1 | 100.00 | Using index; Using filesort |
73 +---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
74 1 row in set, 1 warning (0.00 sec)

```

## 2、group by 不再隐式排序

mysql 8.0 对于group by 字段不再隐式排序，如需要排序，必须显式加上order by 子句。

```

1 # ===MySQL 5.7演示===
2 mysql> select count(*),c2 from t1 group by c2;
3 +-----+-----+
4 | count(*) | c2 |
5 +-----+-----+
6 | 1 | 10 |
7 | 2 | 50 |
8 | 1 | 80 |
9 | 1 | 100 |
10 +-----+-----+
11 4 rows in set (0.00 sec)
12
13
14 # ===MySQL 8.0演示===
15 mysql> select count(*),c2 from t1 group by c2; --8.0版本group by不再默认排序
16 +-----+-----+
17 | count(*) | c2 |
18 +-----+-----+
19 | 1 | 10 |
20 | 2 | 50 |
21 | 1 | 100 |
22 | 1 | 80 |
23 +-----+-----+
24 4 rows in set (0.00 sec)
25
26 mysql> select count(*),c2 from t1 group by c2 order by c2; --8.0版本group by不再默认排序，需要自己加order by
27 +-----+-----+
28 | count(*) | c2 |
29 +-----+-----+
30 | 1 | 10 |
31 | 2 | 50 |
32 | 1 | 80 |
33 | 1 | 100 |
34 +-----+-----+
35 4 rows in set (0.00 sec)

```

## 3、增加隐藏索引

使用 invisible 关键字在创建表或者进行表变更中设置索引为隐藏索引。索引隐藏只是不可见，但是数据库后台还是会维护隐藏索引的，在查询时优化器不使用该索引，即使用force index，优化器也不会使用该索引，同时优化器也不会报索引不存在的错误，因为索引仍然真实存在，必要时，也可以把隐藏索引快速恢复成可见。注意，主键不能设置为 invisible。

软删除就可以使用隐藏索引，比如我们觉得某个索引没用了，删除后发现这个索引在某些时候还是有用的，于是又得把这个索引加回来，如果表数据量很大的话，这种操作耗费时间是很多的，成本很高，这时，我们可以将索引先设置为隐藏索引，等到真的

确认索引没用了再删除。

```
1 # 创建t2表，里面的c2字段为隐藏索引
2 mysql> create table t2(c1 int, c2 int, index idx_c1(c1), index idx_c2(c2) invisible);
3 Query OK, 0 rows affected (0.02 sec)
4
5 mysql> show index from t2\G
6 ***** 1. row *****
7 Table: t2
8 Non_unique: 1
9 Key_name: idx_c1
10 Seq_in_index: 1
11 Column_name: c1
12 Collation: A
13 Cardinality: 0
14 Sub_part: NULL
15 Packed: NULL
16 Null: YES
17 Index_type: BTREE
18 Comment:
19 Index_comment:
20 Visible: YES
21 Expression: NULL
22 ***** 2. row *****
23 Table: t2
24 Non_unique: 1
25 Key_name: idx_c2
26 Seq_in_index: 1
27 Column_name: c2
28 Collation: A
29 Cardinality: 0
30 Sub_part: NULL
31 Packed: NULL
32 Null: YES
33 Index_type: BTREE
34 Comment:
35 Index_comment:
36 Visible: NO --隐藏索引不可见
37 Expression: NULL
38 2 rows in set (0.00 sec)
39
40 mysql> explain select * from t2 where c1=1;
41 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
42 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
43 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
44 | 1 | SIMPLE | t2 | NULL | ref | idx_c1 | idx_c1 | 5 | const | 1 | 100.00 | NULL |
45 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
46 1 row in set, 1 warning (0.00 sec)
47
48 mysql> explain select * from t2 where c2=1; --隐藏索引c2不会被使用
49 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
50 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
```

```

51 +-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
52 | 1 | SIMPLE | t2 | NULL | ALL | NULL | NULL | NULL | NULL | 1 | 100.00 | Using where |
53 +-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
54 1 row in set, 1 warning (0.00 sec)
55
56 mysql> select @@optimizer_switch\G --查看各种参数
57 ***** 1. row *****
58 @@optimizer_switch:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_nested_loop=on,batched_key_access=off,materialization=on,semijoin=on,loosescan=on,firstmatch=on,duplicateweedout=on,subquery_materialization_cost_based=on,use_index_extensions=on,condition_fanout_filter=on,derived_merge=on,use_invisible_indexes=off,skip_scan=on,hash_join=on
59 1 row in set (0.00 sec)
60
61 mysql> set session optimizer_switch="use_invisible_indexes=on"; ----在会话级别设置查询优化器可以看到隐藏索引
62 Query OK, 0 rows affected (0.00 sec)
63
64 mysql> select @@optimizer_switch\G
65 ***** 1. row *****
66 @@optimizer_switch:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_nested_loop=on,batched_key_access=off,materialization=on,semijoin=on,loosescan=on,firstmatch=on,duplicateweedout=on,subquery_materialization_cost_based=on,use_index_extensions=on,condition_fanout_filter=on,derived_merge=on,use_invisible_indexes=on,skip_scan=on,hash_join=on
67 1 row in set (0.00 sec)
68
69 mysql> explain select * from t2 where c2=1;
70 +-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
71 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
72 +-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
73 | 1 | SIMPLE | t2 | NULL | ref | idx_c2 | idx_c2 | 5 | const | 1 | 100.00 | NULL |
74 +-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
75 1 row in set, 1 warning (0.00 sec)
76
77 mysql> alter table t2 alter index idx_c2 visible;
78 Query OK, 0 rows affected (0.02 sec)
79 Records: 0 Duplicates: 0 Warnings: 0
80
81 mysql> alter table t2 alter index idx_c2 invisible;
82 Query OK, 0 rows affected (0.01 sec)
83 Records: 0 Duplicates: 0 Warnings: 0
84

```

#### 4、新增函数索引

之前我们知道，如果在查询中加入了函数，索引不生效，所以MySQL 8引入了函数索引，MySQL 8.0.13开始支持在索引中使用函数(表达式)的值。

函数索引基于虚拟列功能实现，在MySQL中相当于新增了一个列，这个列会根据你的函数来进行计算结果，然后使用函数索引的时候就会用这个计算后的列作为索引。

```

1 mysql> create table t3(c1 varchar(10),c2 varchar(10));
2 Query OK, 0 rows affected (0.02 sec)
3
4 mysql> create index idx_c1 on t3(c1); --创建普通索引

```

```

5 Query OK, 0 rows affected (0.03 sec)
6 Records: 0 Duplicates: 0 Warnings: 0
7
8 mysql> create index func_idx on t3((UPPER(c2))); --创建一个大写的函数索引
9 Query OK, 0 rows affected (0.03 sec)
10 Records: 0 Duplicates: 0 Warnings: 0
11
12 mysql> show index from t3\G
13 ***** 1. row *****
14 Table: t3
15 Non_unique: 1
16 Key_name: idx_c1
17 Seq_in_index: 1
18 Column_name: c1
19 Collation: A
20 Cardinality: 0
21 Sub_part: NULL
22 Packed: NULL
23 Null: YES
24 Index_type: BTREE
25 Comment:
26 Index_comment:
27 Visible: YES
28 Expression: NULL
29 ***** 2. row *****
30 Table: t3
31 Non_unique: 1
32 Key_name: func_idx
33 Seq_in_index: 1
34 Column_name: NULL
35 Collation: A
36 Cardinality: 0
37 Sub_part: NULL
38 Packed: NULL
39 Null: YES
40 Index_type: BTREE
41 Comment:
42 Index_comment:
43 Visible: YES
44 Expression: upper(`c2`) --函数表达式
45 2 rows in set (0.00 sec)
46
47 mysql> explain select * from t3 where upper(c1)='ZHUGE' ;
48 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
49 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
50 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
51 | 1 | SIMPLE | t3 | NULL | ALL | NULL | NULL | NULL | NULL | NULL | 1 | 100.00 | Using where |
52 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
53 1 row in set, 1 warning (0.00 sec)
54
55 mysql> explain select * from t3 where upper(c2)='ZHUGE'; --使用了函数索引
56 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

```

57 | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Ex
tra |
58 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
59 | 1 | SIMPLE | t3 | NULL | ref | func_idx | func_idx | 43 | const | 1 | 100.00 | NULL |
60 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
61 1 row in set, 1 warning (0.00 sec)
62

```

## 5、innodb存储引擎select for update跳过锁等待

对于select ... for share(8.0新增加查询共享锁的语法)或select ... for update，在语句后面添加NOWAIT、SKIP LOCKED语法可以跳过锁等待，或者跳过锁定。

在5.7及之前的版本，select...for update，如果获取不到锁，会一直等待，直到innodb\_lock\_wait\_timeout超时。

在8.0版本，通过添加nowait，skip locked语法，能够立即返回。如果查询的行已经加锁，那么nowait会立即报错返回，而skip locked也会立即返回，只是返回的结果中不包含被锁定的行。

应用场景比如查询余票记录，如果某些记录已经被锁定，用skip locked可以跳过被锁定的记录，只返回没有锁定的记录，提高系统性能。

```

1 # 先打开一个session1:
2 mysql> select * from t1;
3 +-----+-----+
4 | c1 | c2 |
5 +-----+-----+
6 | 1 | 10 |
7 | 2 | 50 |
8 | 3 | 50 |
9 | 4 | 100 |
10 | 5 | 80 |
11 +-----+-----+
12 5 rows in set (0.00 sec)
13
14 mysql> begin;
15 Query OK, 0 rows affected (0.00 sec)
16
17 mysql> update t1 set c2 = 60 where c1 = 2; --锁定第二条记录
18 Query OK, 1 row affected (0.00 sec)
19 Rows matched: 1 Changed: 1 Warnings: 0
20
21
22 # 另外一个session2:
23 mysql> select * from t1 where c1 = 2 for update; --等待超时
24 ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
25
26 mysql> select * from t1 where c1 = 2 for update nowait; --查询立即返回
27 ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set.
28
29 mysql> select * from t1 for update skip locked; --查询立即返回，过滤掉了第二行记录
30 +-----+-----+
31 | c1 | c2 |
32 +-----+-----+
33 | 1 | 10 |
34 | 3 | 50 |
35 | 4 | 100 |
36 | 5 | 80 |
37 +-----+-----+
38 4 rows in set (0.00 sec)

```



## 6、新增innodb\_dedicated\_server自适应参数

能够让InnoDB根据服务器上检测到的内存大小自动配置innodb\_buffer\_pool\_size, innodb\_log\_file\_size等参数, 会尽可能多的占用系统可占用资源提升性能。解决非专业人员安装数据库后默认初始化数据库参数默认值偏低的问题, 前提是服务器是专用来给MySQL数据库的, 如果还有其他软件或者资源或者多实例MySQL使用, 不建议开启该参数, 不然会影响其它程序。

```
1 mysql> show variables like '%innodb_dedicated_server%'; --默认是OFF关闭, 修改为ON打开
2 +-----+
3 | Variable_name | Value |
4 +-----+
5 | innodb_dedicated_server | OFF |
6 +-----+
7 1 row in set (0.02 sec)
8
```

## 7、死锁检查控制

MySQL 8.0 (MySQL 5.7.15) 增加了一个新的动态变量 innodb\_deadlock\_detect, 用于控制系统是否执行 InnoDB 死锁检查, 默认是打开的。死锁检测会耗费数据库性能的, 对于高开发的系统, 我们可以关闭死锁检测功能, 提高系统性能。但是我们要确保系统极少情况会发生死锁, 同时要将锁等待超时参数调小一点, 以防出现死锁等待过久的情况。

```
1 mysql> show variables like '%innodb_deadlock_detect%'; --默认是打开的
2 +-----+
3 | Variable_name | Value |
4 +-----+
5 | innodb_deadlock_detect | ON |
6 +-----+
7 1 row in set, 1 warning (0.01 sec)
```

## 8、undo文件不再使用系统表空间

默认创建2个UNDO表空间, 不再使用系统表空间。

```
[root@localhost mysql]# ls
auto.cnf      client-key.pem  ibdata1        #innodb_temp  mysql-bin.000011  mysql-bin.000015  mysql.ibd      private_key.pem  sys
ca-key.pem    #ib_16384_0.dblwr  ib_logfile0    mysql         mysql-bin.000012  mysql-bin.000016  mysql.log      public_key.pem   test
ca.pem        #ib_16384_1.dblwr  ib_logfile1    mysql-bin.000009  mysql-bin.000013  mysql-bin.index  mysql.pid      server-cert.pem  undo_001
client-cert.pem  ib_buffer_pool  ibtmp1         mysql-bin.000010  mysql-bin.000014  mysql-d-auto.cnf  performance_schema  server-key.pem  undo_002
```

## 9、binlog日志过期时间精确到秒

之前是天, 并且参数名称发生变化. 在8.0版本之前, binlog日志过期时间设置都是设置expire\_logs\_days参数, 而在8.0版本中, MySQL默认使用binlog\_expire\_logs\_seconds参数。

## 10、窗口函数(Window Functions): 也称分析函数

从 MySQL 8.0 开始, 新增了一个叫窗口函数的概念, 它可以用来实现若干新的查询方式。窗口函数与 SUM()、COUNT() 这种分组聚合函数类似, 在聚合函数后面加上over()就变成窗口函数了, 在括号里可以加上partition by等分组关键字指定如何分组, 窗口函数即便分组也不会将多行查询结果合并为一行, 而是将结果放回多行当中, 即窗口函数不需要再使用 GROUP BY。

```
1 # 创建一张账户余额表
2 CREATE TABLE `account_channel` (
3   `id` int NOT NULL AUTO_INCREMENT,
4   `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT NULL COMMENT '姓名',
5   `channel` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT NULL COMMENT '账户渠道',
6   `balance` int DEFAULT NULL COMMENT '余额',
7   PRIMARY KEY (`id`)
8 ) ENGINE=InnoDB
9
10 # 插入一些示例数据
11 INSERT INTO `test`.`account_channel` (`id`, `name`, `channel`, `balance`) VALUES ('1', 'zhuge', 'wx', '100');
12 INSERT INTO `test`.`account_channel` (`id`, `name`, `channel`, `balance`) VALUES ('2', 'zhuge', 'alipay', '200');
```

```

13 INSERT INTO `test`.`account_channel` (`id`, `name`, `channel`, `balance`) VALUES ('3', 'zhuge', 'yinhang',
'300');
14 INSERT INTO `test`.`account_channel` (`id`, `name`, `channel`, `balance`) VALUES ('4', 'lilei', 'wx',
'200');
15 INSERT INTO `test`.`account_channel` (`id`, `name`, `channel`, `balance`) VALUES ('5', 'lilei', 'alipay',
'100');
16 INSERT INTO `test`.`account_channel` (`id`, `name`, `channel`, `balance`) VALUES ('6', 'hanmeimei', 'wx',
'500');

```

17

```
18 mysql> select * from account_channel;
```

```

19 +---+-----+-----+-----+
20 | id | name | channel | balance |
21 +---+-----+-----+-----+
22 | 1 | zhuge | wx | 100 |
23 | 2 | zhuge | alipay | 200 |
24 | 3 | zhuge | yinhang | 300 |
25 | 4 | lilei | wx | 200 |
26 | 5 | lilei | alipay | 100 |
27 | 6 | hanmeimei | wx | 500 |

```

```
28 +---+-----+-----+-----+
```

```
29 6 rows in set (0.00 sec)
```

30

```
31 mysql> select name,sum(balance) from account_channel group by name;
```

```

32 +-----+-----+
33 | name | sum(balance) |
34 +-----+-----+
35 | zhuge | 600 |
36 | lilei | 300 |
37 | hanmeimei | 500 |

```

```
38 +-----+-----+
```

```
39 3 rows in set (0.00 sec)
```

40

41 # 在聚合函数后面加上over()就变成分析函数了，后面可以不用再加group by制定分组，因为在over里已经用partition关键字指明了如何分组计算，这种可以保留原有表数据的结构，不会像分组聚合函数那样每组只返回一条数据

```
42 mysql> select name,channel,balance,sum(balance) over(partition by name) as sum_balance from
account_channel;
```

```

43 +-----+-----+-----+-----+
44 | name | channel | balance | sum_balance |
45 +-----+-----+-----+-----+
46 | hanmeimei | wx | 500 | 500 |
47 | lilei | wx | 200 | 300 |
48 | lilei | alipay | 100 | 300 |
49 | zhuge | wx | 100 | 600 |
50 | zhuge | alipay | 200 | 600 |
51 | zhuge | yinhang | 300 | 600 |

```

```
52 +-----+-----+-----+-----+
```

```
53 6 rows in set (0.00 sec)
```

54

```
55 mysql> select name,channel,balance,sum(balance) over(partition by name order by balance) as sum_balance from
m account_channel;
```

```

56 +-----+-----+-----+-----+
57 | name | channel | balance | sum_balance |
58 +-----+-----+-----+-----+
59 | hanmeimei | wx | 500 | 500 |
60 | lilei | alipay | 100 | 100 |
61 | lilei | wx | 200 | 300 |
62 | zhuge | wx | 100 | 100 |
63 | zhuge | alipay | 200 | 300 |

```

```

64 | zhuge | yinhang | 300 | 600 |
65 +-----+-----+-----+-----+
66 6 rows in set (0.00 sec)
67
68
69 # over()里如果不加条件，则默认使用整个表的数据做运算
70 mysql> select name,channel,balance,sum(balance) over() as sum_balance from account_channel;
71 +-----+-----+-----+-----+
72 | name | channel | balance | sum_balance |
73 +-----+-----+-----+-----+
74 | zhuge | wx | 100 | 1400 |
75 | zhuge | alipay | 200 | 1400 |
76 | zhuge | yinhang | 300 | 1400 |
77 | lilei | wx | 200 | 1400 |
78 | lilei | alipay | 100 | 1400 |
79 | hanmeimei | wx | 500 | 1400 |
80 +-----+-----+-----+-----+
81 6 rows in set (0.00 sec)
82
83 mysql> select name,channel,balance,avg(balance) over(partition by name) as avg_balance from
account_channel;
84 +-----+-----+-----+-----+
85 | name | channel | balance | avg_balance |
86 +-----+-----+-----+-----+
87 | hanmeimei | wx | 500 | 500.0000 |
88 | lilei | wx | 200 | 150.0000 |
89 | lilei | alipay | 100 | 150.0000 |
90 | zhuge | wx | 100 | 200.0000 |
91 | zhuge | alipay | 200 | 200.0000 |
92 | zhuge | yinhang | 300 | 200.0000 |
93 +-----+-----+-----+-----+
94 6 rows in set (0.00 sec)

```

#### 专用窗口函数：

- 序号函数：ROW\_NUMBER()、RANK()、DENSE\_RANK()
- 分布函数：PERCENT\_RANK()、CUME\_DIST()
- 前后函数：LAG()、LEAD()
- 头尾函数：FIRST\_VALUE()、LAST\_VALUE()
- 其它函数：NTH\_VALUE()、NTILE()

```

1 # 按照balance字段排序，展示序号
2 mysql> select name,channel,balance,row_number() over(order by balance) as row_number1 from account_channel;
3 +-----+-----+-----+-----+
4 | name | channel | balance | row_number1 |
5 +-----+-----+-----+-----+
6 | zhuge | wx | 100 | 1 |
7 | lilei | alipay | 100 | 2 |
8 | zhuge | alipay | 200 | 3 |
9 | lilei | wx | 200 | 4 |
10 | zhuge | yinhang | 300 | 5 |
11 | hanmeimei | wx | 500 | 6 |
12 +-----+-----+-----+-----+
13 6 rows in set (0.00 sec)
14
15 # 按照balance字段排序，first_value()选出排第一的余额
16 mysql> select name,channel,balance,first_value(balance) over(order by balance) as first1 from account_chann
el;

```

```

17 +-----+-----+-----+-----+
18 | name | channel | balance | first1 |
19 +-----+-----+-----+-----+
20 | zhuge | wx | 100 | 100 |
21 | lilei | alipay | 100 | 100 |
22 | zhuge | alipay | 200 | 100 |
23 | lilei | wx | 200 | 100 |
24 | zhuge | yinhang | 300 | 100 |
25 | hanmeimei | wx | 500 | 100 |
26 +-----+-----+-----+-----+
27 6 rows in set (0.01 sec)

```

## 12、默认字符集由latin1变为utf8mb4

在8.0版本之前，默认字符集为latin1，utf8指向的是utf8mb3，8.0版本默认字符集为utf8mb4，utf8默认指向的也是utf8mb4。

## 13、MyISAM系统表全部换成InnoDB表

将系统表(mysql)和数据字典表全部改为InnoDB存储引擎，默认的MySQL实例将不包含MyISAM表，除非手动创建MyISAM表。

## 14、元数据存储变动

MySQL 8.0删除了之前版本的元数据文件，例如表结构.frm等文件，全部集中放入mysql.ibd文件里。可以看见下图test库文件夹里已经没有了frm文件。

```

[root@localhost mysql]# ls test
account.ibd  employee.ibd  film.ibd  t1.ibd  test_mysam_356.sdi  test_mysam.MYI
actor.ibd  film_actor.ibd  leaf_id.ibd  test_innodb.ibd  test_mysam.MYD  test_order_id.ibd
[root@localhost mysql]# ls
auto.cnf  client-key.pem  ibdata1  #innodb_temp  mysql-bin.000011  mysql-bin.000015  mysql.ibd  private_key.pem  sys
ca-key.pem  #ib_16384.0.dblwr  ib_logfile0  mysql  mysql-bin.000012  mysql-bin.000016  mysql.log  public_key.pem  test
ca.pem  #ib_16384.1.dblwr  ib_logfile1  mysql-bin.000009  mysql-bin.000013  mysql-bin.index  mysql.pid  server-cert.pem  undo_001
client-cert.pem  ib_buffer_pool  ibtmp1  mysql-bin.000010  mysql-bin.000014  mysqlid-auto.cnf  performance_schema  server-key.pem  undo_002

```

## 15、自增变量持久化

在8.0之前的版本，自增主键AUTO\_INCREMENT的值如果大于max(primary key)+1，在MySQL重启后，会重置AUTO\_INCREMENT=max(primary key)+1，这种现象在某些情况下会导致业务主键冲突或者其他难以发现的问题。自增主键重启重置的问题很早就被发现(<https://bugs.mysql.com/bug.php?id=199>)，一直到8.0才被解决，8.0版本将会对AUTO\_INCREMENT值进行持久化，MySQL重启后，该值将不会改变。

```

1 # ===MySQL 5.7演示===
2 mysql> create table t(id int auto_increment primary key,c1 varchar(20));
3 Query OK, 0 rows affected (0.03 sec)
4
5 mysql> insert into t(c1) values('zhuge1'),('zhuge2'),('zhuge3');
6 Query OK, 3 rows affected (0.00 sec)
7 Records: 3 Duplicates: 0 Warnings: 0
8
9 mysql> select * from t;
10 +-----+
11 | id | c1 |
12 +-----+
13 | 1 | zhuge1 |
14 | 2 | zhuge2 |
15 | 3 | zhuge3 |
16 +-----+
17 3 rows in set (0.00 sec)
18
19 mysql> delete from t where id = 3;
20 Query OK, 1 row affected (0.01 sec)
21
22 mysql> select * from t;

```

```

23 +-----+
24 | id | c1 |
25 +-----+
26 | 1 | zhuge1 |
27 | 2 | zhuge2 |
28 +-----+
29 2 rows in set (0.00 sec)
30
31 mysql> exit;
32 Bye
33
34 # 重启MySQL服务，并重新连接MySQL
35 mysql> insert into t(c1) values('zhuge4');
36 Query OK, 1 row affected (0.01 sec)
37
38 mysql> select * from t;
39 +-----+
40 | id | c1 |
41 +-----+
42 | 1 | zhuge1 |
43 | 2 | zhuge2 |
44 | 3 | zhuge4 |
45 +-----+
46 3 rows in set (0.00 sec)
47
48 mysql> update t set id = 5 where c1 = 'zhuge1';
49 Query OK, 1 row affected (0.01 sec)
50 Rows matched: 1 Changed: 1 Warnings: 0
51
52 mysql> select * from t;
53 +-----+
54 | id | c1 |
55 +-----+
56 | 2 | zhuge2 |
57 | 3 | zhuge4 |
58 | 5 | zhuge1 |
59 +-----+
60 3 rows in set (0.00 sec)
61
62 mysql> insert into t(c1) values('zhuge5');
63 Query OK, 1 row affected (0.01 sec)
64
65 mysql> select * from t;
66 +-----+
67 | id | c1 |
68 +-----+
69 | 2 | zhuge2 |
70 | 3 | zhuge4 |
71 | 4 | zhuge5 |
72 | 5 | zhuge1 |
73 +-----+
74 4 rows in set (0.00 sec)
75
76 mysql> insert into t(c1) values('zhuge6');
77 ERROR 1062 (23000): Duplicate entry '5' for key 'PRIMARY'
78
79

```

```

80
81 # ===MySQL 8.0演示===
82 mysql> create table t(id int auto_increment primary key,c1 varchar(20));
83 Query OK, 0 rows affected (0.02 sec)
84
85 mysql> insert into t(c1) values('zhuge1'),('zhuge2'),('zhuge3');
86 Query OK, 3 rows affected (0.00 sec)
87 Records: 3 Duplicates: 0 Warnings: 0
88
89 mysql> select * from t;
90 +-----+
91 | id | c1 |
92 +-----+
93 | 1 | zhuge1 |
94 | 2 | zhuge2 |
95 | 3 | zhuge3 |
96 +-----+
97 3 rows in set (0.00 sec)
98
99 mysql> delete from t where id = 3;
100 Query OK, 1 row affected (0.01 sec)
101
102 mysql> select * from t;
103 +-----+
104 | id | c1 |
105 +-----+
106 | 1 | zhuge1 |
107 | 2 | zhuge2 |
108 +-----+
109 2 rows in set (0.00 sec)
110
111 mysql> exit;
112 Bye
113 [root@localhost ~]# service mysqld restart
114 Shutting down MySQL.... SUCCESS!
115 Starting MySQL... SUCCESS!
116
117 # 重新连接MySQL
118 mysql> insert into t(c1) values('zhuge4');
119 Query OK, 1 row affected (0.00 sec)
120
121 mysql> select * from t; --生成的id为4, 不是3
122 +-----+
123 | id | c1 |
124 +-----+
125 | 1 | zhuge1 |
126 | 2 | zhuge2 |
127 | 4 | zhuge4 |
128 +-----+
129 3 rows in set (0.00 sec)
130
131 mysql> update t set id = 5 where c1 = 'zhuge1';
132 Query OK, 1 row affected (0.01 sec)
133 Rows matched: 1 Changed: 1 Warnings: 0
134
135 mysql> select * from t;
136 +-----+

```

```

137 | id | c1 |
138 +-----+
139 | 2 | zhuge2 |
140 | 4 | zhuge4 |
141 | 5 | zhuge1 |
142 +-----+
143 3 rows in set (0.00 sec)
144
145 mysql> insert into t(c1) values('zhuge5');
146 Query OK, 1 row affected (0.00 sec)
147
148 mysql> select * from t;
149 +-----+
150 | id | c1 |
151 +-----+
152 | 2 | zhuge2 |
153 | 4 | zhuge4 |
154 | 5 | zhuge1 |
155 | 6 | zhuge5 |
156 +-----+
157 4 rows in set (0.00 sec)

```

## 16、DDL原子化

InnoDB表的DDL支持事务完整性，要么成功要么回滚。

MySQL 8.0 开始支持原子 DDL 操作，其中与表相关的原子 DDL 只支持 InnoDB 存储引擎。一个原子 DDL 操作内容包括：更新数据字典，存储引擎层的操作，在 binlog 中记录 DDL 操作。支持与表相关的 DDL：数据库、表空间、表、索引的 CREATE、ALTER、DROP 以及 TRUNCATE TABLE。支持的其它 DDL：存储程序、触发器、视图、UDF 的 CREATE、DROP 以及 ALTER 语句。支持账户管理相关的 DDL：用户和角色的 CREATE、ALTER、DROP 以及适用的 RENAME等等。

```

1 # MySQL 5.7
2 mysql> show tables;
3 +-----+
4 | Tables_in_test |
5 +-----+
6 | account |
7 | actor |
8 | employee |
9 | film |
10 | film_actor |
11 | leaf_id |
12 | t1 |
13 | test_innodb |
14 | test_myisam |
15 | test_order_id |
16 +-----+
17 10 rows in set (0.01 sec)
18
19 mysql> drop table t1,t2; //删除表报错不会回滚，t1表会被删除
20 ERROR 1051 (42S02): Unknown table 'test.t2'
21 mysql> show tables;
22 +-----+
23 | Tables_in_test |
24 +-----+
25 | account |
26 | actor |
27 | employee |

```

```

28 | film |
29 | film_actor |
30 | leaf_id |
31 | test_innodb |
32 | test_myisam |
33 | test_order_id |
34 +-----+
35 9 rows in set (0.00 sec)
36
37
38 # MySQL 8.0
39 mysql> show tables;
40 +-----+
41 | Tables_in_test |
42 +-----+
43 | account |
44 | actor |
45 | employee |
46 | film |
47 | film_actor |
48 | leaf_id |
49 | t1 |
50 | test_innodb |
51 | test_myisam |
52 | test_order_id |
53 +-----+
54 10 rows in set (0.00 sec)
55
56 mysql> drop table t1,t2; //删除表报错会回滚，t1表依然在
57 ERROR 1051 (42S02): Unknown table 'test.t2'
58 mysql> show tables;
59 +-----+
60 | Tables_in_test |
61 +-----+
62 | account |
63 | actor |
64 | employee |
65 | film |
66 | film_actor |
67 | leaf_id |
68 | t1 |
69 | test_innodb |
70 | test_myisam |
71 | test_order_id |
72 +-----+
73 10 rows in set (0.00 sec)

```

## 17、参数修改持久化

MySQL 8.0版本支持在线修改全局参数并持久化，通过加上PERSIST关键字，可以将修改的参数持久化到新的配置文件（mysqld-auto.cnf）中，重启MySQL时，可以从该配置文件获取到最新的配置参数。set global 设置的变量参数在mysql重启后会失效。

```

1 mysql> set persist innodb_lock_wait_timeout=25;
2 系统会在数据目录下生成一个包含json格式的mysqld-auto.cnf 的文件，格式化后如下所示，当my.cnf 和mysqld-auto.cnf 同时
  存在时，后者具有更高优先级。
3 {
4   "Version": 1,

```



```
5  "mysql_server": {  
6    "innodb_lock_wait_timeout": {  
7      "Value": "25",  
8      "Metadata": {  
9        "Timestamp": 1675290252103863,  
10       "User": "root",  
11       "Host": "localhost"  
12     }  
13   }  
14 }  
15 }
```

1 文档: [08-VIP-Mysql全局优化与Mysql 8.0新特详解](#)

2 链接: <http://note.youdao.com/noteshare?id=c0b91c9b60a787a2293a780ca5d3c151&sub=A8EEC383AEEB4D7C883228E7CAF37B17>