# MySQL数据库开发设计规范

此规范为规范开发人员如何基于ORACLE MYSQL开发应用和使用的开发规范,当前还在持续完善中。

## 默认MYSQL存储引擎

|  |  |
| --- | --- |
| 要求内容 | MySQL支持INNODB,MYISAM等数据库引擎，INNODB引擎是MySQL5.5版本以后的默认引擎,MYSQL 5.7开始系统表也统一改为INNODB引擎,INNODB支持事务,行级锁,有更好的数据恢复能力、更好的并发性能,同时对多核CPU、大内存、SSD等硬件支持更好,支持数据联机热备份等,因此INNODB相比MyISAM有明显优势，尤其在金融行业。  **【规范】存储引擎必须使用InnoDB，使用其他存储引擎需特别申请。**  运行如下命令修改：在cnf文件中加入以下内容  [mysqld]  default-storage-engine = innodb  检查  mysql> show global variables like '%storage%';  +----------------------------+--------+  | Variable\_name | Value |  +----------------------------+--------+  | default\_storage\_engine | InnoDB |  | default\_tmp\_storage\_engine | InnoDB |  | storage\_engine | InnoDB |  +----------------------------+--------+  3 rows in set (0.01 sec)  mysql> |

## 字符集使用UTF8

|  |  |
| --- | --- |
| 要求内容 | 绝大多数数据库产品UTF8字符集存储汉字占用3个字节,存储英文字符占用1个字节,而MYSQL默认按字符方式存储数据,对于汉字，例如varchar(10)可以存10个汉字.所以对于从其它数据库产品迁移到MYSQL数据库UTF8字符集情况下无需担心字符集转换问题。  UTF8统一而且通用,不会出现转码出现乱码风险。  **【规范】所有字符集参数均设置为UTF8**  运行如下命令修改：在cnf文件中加入以下内容  [mysqld]  character-set-server = utf8  mysql> show variables like '%character%';  +--------------------------+-----------------------------------------------------------+  | Variable\_name | Value |  +--------------------------+-----------------------------------------------------------+  | character\_set\_client | latin1 |  | character\_set\_connection | latin1 |  | character\_set\_database | utf8 |  | character\_set\_filesystem | binary |  | character\_set\_results | latin1 |  | character\_set\_server | utf8 |  | character\_set\_system | utf8 |  | character\_sets\_dir | /data/MySQL/mysql/mariadb/mariadb-10.0.10/share/charsets/ |  +--------------------------+-----------------------------------------------------------+  8 rows in set (0.00 sec)  mysql> SET character\_set\_client = utf8;  Query OK, 0 rows affected (0.00 sec)  mysql> SET character\_set\_connection = utf8;  Query OK, 0 rows affected (0.00 sec)  mysql> SET character\_set\_results = utf8;  Query OK, 0 rows affected (0.00 sec)  mysql> show variables like '%character%';  +--------------------------+-----------------------------------------------------------+  | Variable\_name | Value |  +--------------------------+-----------------------------------------------------------+  | character\_set\_client | utf8 |  | character\_set\_connection | utf8 |  | character\_set\_database | utf8 |  | character\_set\_filesystem | binary |  | character\_set\_results | utf8 |  | character\_set\_server | utf8 |  | character\_set\_system | utf8 |  | character\_sets\_dir | /data/MySQL/mysql/mariadb/mariadb-10.0.10/share/charsets/ |  +--------------------------+-----------------------------------------------------------+  8 rows in set (0.00 sec) |

## 事务隔离级

|  |  |
| --- | --- |
| 要求内容 | MYSQL默认支持如下4中事务隔离级：  **Read Uncommitted(读取未提交内容)**  所有事务都可以看到其他未提交事务的执行结果,该隔离级别很少用于实际应用。读取未提交的数据，也被称之为脏读（Dirty Read）.  分别在A、B两个客户端执行：  session A：  root@(none) 10:54>SET GLOBAL tx\_isolation='READ-UNCOMMITTED';  Query OK, 0 rows affected (0.00 sec)  root@(none) 10:54>SELECT @@tx\_isolation;  +------------------+  | @@tx\_isolation |  +------------------+  | READ-UNCOMMITTED |  +------------------+  1 row in set (0.00 sec)  root@(none) 10:54>use test;  Database changed  root@test 10:55>begin;  Query OK, 0 rows affected (0.00 sec)  root@test 10:55>select \* from test1;  +------+  | a |  +------+  | 1 |  | 2 |  | 3 |  | 4 |  +------+  4 rows in set (0.00 sec)  session B：  root@test 10:58>select @@tx\_isolation;  +------------------+  | @@tx\_isolation |  +------------------+  | READ-UNCOMMITTED |  +------------------+  1 row in set (0.00 sec)  root@test 10:58>  root@test 10:58>begin;  Query OK, 0 rows affected (0.00 sec)  root@test 10:58>insert into test.test1 values (999);  Query OK, 1 row affected (0.00 sec)  root@test 10:58>select \* from test.test1;  +------+  | a |  +------+  | 1 |  | 2 |  | 3 |  | 4 |  | 999 |  +------+  5 rows in set (0.00 sec)  此处B客户端并未commit；  再查看A客户端：  root@test 10:58>select \* from test1;  +------+  | a |  +------+  | 1 |  | 2 |  | 3 |  | 4 |  | 999 |  +------+  5 rows in set (0.00 sec)  此处A可以看到新的记录。  **Read Committed(读取提交内容)**  这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。事务只能看见已经提交事务所做的改变。  session A：  root@(none) 11:10>SET GLOBAL tx\_isolation='READ-COMMITTED';  Query OK, 0 rows affected (0.00 sec)  root@(none) 11:10>SELECT @@tx\_isolation;  +----------------+  | @@tx\_isolation |  +----------------+  | READ-COMMITTED |  +----------------+  1 row in set (0.00 sec)  root@(none) 11:10>  root@(none) 11:10>begin;  Query OK, 0 rows affected (0.00 sec)  root@(none) 11:10>select \* from test.test1;  +------+  | a |  +------+  | 1 |  | 2 |  | 3 |  | 4 |  +------+  session B：  root@test 11:11>begin;  Query OK, 0 rows affected (0.00 sec)  root@test 11:11>select \* from test.test1;  +------+  | a |  +------+  | 1 |  | 2 |  | 3 |  | 4 |  +------+  4 rows in set (0.00 sec)  root@test 11:11>  root@test 11:11>delete from test.test1 where a=1;  Query OK, 1 row affected (0.00 sec)  root@test 11:12>select \* from test.test1;  +------+  | a |  +------+  | 2 |  | 3 |  | 4 |  +------+  此时查询A客户端：  root@(none) 11:12>select \* from test.test1;  +------+  | a |  +------+  | 1 |  | 2 |  | 3 |  | 4 |  +------+  此处看出A客户端无变化，在B客户端执行commit后再查看A客户端：  root@(none) 11:13>select \* from test.test1;  +------+  | a |  +------+  | 2 |  | 3 |  | 4 |  +------+  可以看到A客户端的数据已经变了。已提交读只允许读取已提交的记录，但不要求可重复读。  用MVCC来说就是读取当前行的最新版本。  **Repeatable Read(可重读)**  这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。  session A：  root@(none) 11:17>SET GLOBAL tx\_isolation='REPEATABLE-READ';  Query OK, 0 rows affected (0.00 sec)  root@(none) 11:17>  root@(none) 11:17>  root@(none) 11:17>SELECT @@tx\_isolation;  +-----------------+  | @@tx\_isolation |  +-----------------+  | REPEATABLE-READ |  +-----------------+  1 row in set (0.00 sec)  root@(none) 11:17>BEGIN; ----> 这里非常重要,开启一个事务  Query OK, 0 rows affected (0.00 sec)  session B：  root@test 11:20>select @@tx\_isolation;  +-----------------+  | @@tx\_isolation |  +-----------------+  | REPEATABLE-READ |  +-----------------+  1 row in set (0.00 sec)  root@test 11:20>insert into test.test1 values (555);  Query OK, 1 row affected (0.00 sec)  root@test 11:20>commit;  Query OK, 0 rows affected (0.00 sec)  root@test 11:21>  root@test 11:21>select \* from test.test1;  +------+  | a |  +------+  | 2 |  | 3 |  | 4 |  | 555 |  +------+  4 rows in set (0.00 sec)  此处在B客户端上已经commit.  然后查看A客户端：  root@(none) 11:22>SELECT \* FROM test.test1;  +------+  | a |  +------+  | 2 |  | 3 |  | 4 |  +------+  3 rows in set (0.00 sec)  root@(none) 11:22>commit;  Query OK, 0 rows affected (0.00 sec)  root@(none) 11:22>SELECT \* FROM test.test1;  +------+  | a |  +------+  | 2 |  | 3 |  | 4 |  | 555 |  +------+  4 rows in set (0.00 sec)  在A客户端上提交后可以看到新数据。  也就是说在可重复读隔离级别只能读取已经提交的数据，并且在一个事务内，读取的数据就是事务开始时的数据。  **Serializable(可串行化)**  这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。该类型在A客户端操作test.test1表时会锁定该数据，如果B客户端想要操作test.test1就需要等待A客户端释放。  **【规范】事务隔离级必须调整为READ COMMITTED.**  MYSQL默认当前的会话事务级别:  mysql> select @@tx\_isolation;  | @@tx\_isolation |  | REPEATABLE-READ |  1 row in set (0.00 sec)  设置事务隔离级方法:  SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED;  SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;  查询事务隔离级别:  SELECT @@global.tx\_isolation;  SELECT @@session.tx\_isolation;  SELECT @@tx\_isolation; |

## 启用事务自提交功能

|  |  |
| --- | --- |
| 要求内容 | **【规范】MYSQL默认事务是自提交的,**  对于应用端事务涉及多条DML的时候,需要强制指定开启事务,当应用端开启事务后,在该事务中所有的SQL会在一个原子级中.  mysql> select @@global.autocommit;  +---------------------+  | @@global.autocommit |  +---------------------+  | 1 |  +---------------------+  1 row in set (0.00 sec)  mysql> select @@session.autocommit;  +----------------------+  | @@session.autocommit |  +----------------------+  | 1 |  +----------------------+  1 row in set (0.00 sec) |

## 分区表分区类型

|  |  |
| --- | --- |
| 要求内容 | MYSQL目前支持range分区,list分区,hash分区，其它分区类型不能使用  举例：range 分区  CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY RANGE (YEAR(separated)) (  PARTITION p0 VALUES LESS THAN (1991),  PARTITION p1 VALUES LESS THAN (1996),  PARTITION p2 VALUES LESS THAN (2001),  PARTITION p3 VALUES LESS THAN MAXVALUE  );  举例：list 分区  CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY LIST(store\_id)  PARTITION pNorth VALUES IN (3,5,6,9,17),  PARTITION pEast VALUES IN (1,2,10,11,19,20),  PARTITION pWest VALUES IN (4,12,13,14,18),  PARTITION pCentral VALUES IN (7,8,15,16)  );  举例：hash 分区,分区数要求为偶数并小于CPU个数  CREATE TABLE employees (  id INT NOT NULL,  fname VARCHAR(30),  lname VARCHAR(30),  hired DATE NOT NULL DEFAULT '1970-01-01',  separated DATE NOT NULL DEFAULT '9999-12-31',  job\_code INT,  store\_id INT  )  PARTITION BY HASH(store\_id)  PARTITIONS 4;  **【规范】考虑到MYSQL对于hash分区对于SELECT操作执行效率没有比较好的帮助，同时MYSQL只支持嵌套循环连接,所以要求只使用range和list分区方式.** |

## 分区表分区索引的注意事项

|  |  |
| --- | --- |
| 要求内容 | 1,mysql分区表的primary key里面的列必须包含用于分区的列,  否则会报错"ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function "  2,如下举例一,创建了11张表，对应11个数据文件，从test1#P#p1.ibd ... test1#P#p11.ibd，一个 test1.frm的逻辑表结构文件，一个test1.par的分区结构文件，在存储引擎层各个分区表之间是完全不知道对方的存在,由上层统一管理,对应用透明. 所以没有全局索引的概念,各个索引都是由innodb针对每个子表独立维护,所以删除某个分区其实将某个分区对应的数据文件和索引文件都删除了.正因为各个子表和索引是完全独立的,所以建表语句中要求主键必须包含分区,否则同一个主键的记录可能会落在多个分区,因此会破坏acid(原子性、[一致性](http://baike.baidu.com/view/552538.htm)、[独立性](http://baike.baidu.com/view/562176.htm)及[持久性](http://baike.baidu.com/view/3849641.htm))中的原子性,所以目前MySQL的分区表要求主键包含所有的的分区字段.  3,表做了分区之后,如果每次sql操作不能具体到某个指定的分区,可能会引起性能极度恶化,OLTP应用必须保证所有的sql操作尽量带上分区字段,并且尽量将需要访问的分区控制在2个之内,否则在大数据情况下引起性能急剧下降,因此我们这边只有流水满足这样的场景.因此使用分区表的时候最好能分析下常用sql针对分区的查询计划(见下面的原理分析).  举例一: 范围分区  mysql> CREATE TABLE test1 (  -> id int(11) NOT NULL AUTO\_INCREMENT,  -> name VARCHAR(20) NOT NULL,  -> submit\_time DATETIME NOT NULL,  -> index time\_index (submit\_time),  -> **primary key (id)** -> )engine innodb charset utf8  -> PARTITION BY RANGE (TO\_DAYS(submit\_time))  -> (  -> PARTITION p1 VALUES LESS THAN (TO\_DAYS('2010-02-01')),  -> PARTITION p2 VALUES LESS THAN (TO\_DAYS('2010-03-01')),  -> PARTITION p3 VALUES LESS THAN (TO\_DAYS('2010-04-01')),  -> PARTITION p4 VALUES LESS THAN (TO\_DAYS('2010-05-01')),  -> PARTITION p5 VALUES LESS THAN (TO\_DAYS('2010-06-01')),  -> PARTITION p6 VALUES LESS THAN (TO\_DAYS('2010-07-01')),  -> PARTITION p7 VALUES LESS THAN (TO\_DAYS('2010-08-01')),  -> PARTITION p8 VALUES LESS THAN (TO\_DAYS('2010-09-01')),  -> PARTITION p9 VALUES LESS THAN (TO\_DAYS('2010-10-01')),  -> PARTITION p10 VALUES LESS THAN (TO\_DAYS('2010-11-01')),  -> PARTITION p11 VALUES LESS THAN (TO\_DAYS('2010-12-01'))  -> );  ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function  mysql> CREATE TABLE test1 (  -> id int(11) NOT NULL AUTO\_INCREMENT,  -> name VARCHAR(20) NOT NULL,  -> submit\_time DATETIME NOT NULL,  -> index time\_index (submit\_time),  -> **primary key (id,submit\_time)** -> )engine innodb charset utf8  -> PARTITION BY RANGE (TO\_DAYS(submit\_time))  -> (  -> PARTITION p1 VALUES LESS THAN (TO\_DAYS('2010-02-01')),  -> PARTITION p2 VALUES LESS THAN (TO\_DAYS('2010-03-01')),  -> PARTITION p3 VALUES LESS THAN (TO\_DAYS('2010-04-01')),  -> PARTITION p4 VALUES LESS THAN (TO\_DAYS('2010-05-01')),  -> PARTITION p5 VALUES LESS THAN (TO\_DAYS('2010-06-01')),  -> PARTITION p6 VALUES LESS THAN (TO\_DAYS('2010-07-01')),  -> PARTITION p7 VALUES LESS THAN (TO\_DAYS('2010-08-01')),  -> PARTITION p8 VALUES LESS THAN (TO\_DAYS('2010-09-01')),  -> PARTITION p9 VALUES LESS THAN (TO\_DAYS('2010-10-01')),  -> PARTITION p10 VALUES LESS THAN (TO\_DAYS('2010-11-01')),  -> PARTITION p11 VALUES LESS THAN (TO\_DAYS('2010-12-01'))  -> );  Query OK, 0 rows affected (0.02 sec)  举例二: 列表分区  mysql> create table category\_part  -> (  -> cid int unsigned not null auto\_increment,  -> cname varchar(64) not null,  -> parent\_id int not null,  -> primary key (cid)  -> )  -> partition by list(parent\_id)  -> (  -> partition p1 values in (1,2,3,6,9),  -> partition p2 values in (4,5,10,22,23),  -> partition p3 values in (7,8,11,12,13),  -> partition p4 values in (14,15,16,17,20),  -> partition p5 values in (18,19,21,24,25)  -> );  ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function  mysql>  mysql> create table category\_part  -> (  -> cid int unsigned not null auto\_increment,  -> cname varchar(64) not null,  -> parent\_id int not null,  -> primary key (cid,parent\_id)  -> )  -> partition by list(parent\_id)  -> (  -> partition p1 values in (1,2,3,6,9),  -> partition p2 values in (4,5,10,22,23),  -> partition p3 values in (7,8,11,12,13),  -> partition p4 values in (14,15,16,17,20),  -> partition p5 values in (18,19,21,24,25)  -> );  Query OK, 0 rows affected (0.01 sec) 举例三:  mysql> select \* from test1;  +----+------+---------------------+  | id | name | submit\_time |  +----+------+---------------------+  | 1 | 1111 | 2010-01-15 00:00:00 |  | 3 | 3333 | 2010-03-15 00:00:00 |  | 4 | 4444 | 2010-04-15 00:00:00 |  | 5 | 5555 | 2010-05-15 00:00:00 |  +----+------+---------------------+  4 rows in set (0.00 sec)  查看涉及分区表的SQL执行计划需要增加partition关键字:  explain partitions select \* from test1 where id=2;  +------+-------------+-------+------------------------------------+------+---------------+---------+---------+-------+------+-- | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | Extra |  +------+-------------+-------+------------------------------------+------+---------------+---------+---------+-------+------+-- | 1 | SIMPLE | test1 | p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11 | ref | PRIMARY | PRIMARY | 4 | const | 11 | |  +------+-------------+-------+------------------------------------+------+---------------+---------+---------+-------+------+--  从这个结果可以发现因为没有带分区字段，通过主键查询其实也查询了11个子表；  查询带分区表的查询计划要增加partitions字段:  explain partitions select \* from test1 where id=2 and submit\_time=date\_format('2010-02-15','%Y-%m-%d');   +------+-------------+-------+------------+-------+--------------------+---------+---------+-------------+------+-------+  | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | Extra |  +------+-------------+-------+------------+-------+--------------------+---------+---------+-------------+------+-------+  | 1 | SIMPLE | test1 | p2 | const | PRIMARY,time\_index | PRIMARY | 12 | const,const | 1 | |  +------+-------------+-------+------------+-------+--------------------+---------+---------+-------------+------+-------+  带上分区字段，就能精确到p2这个分区了.  删除分区其实将数据和索引文件都删除了,如下：  mysql> alter table test1 drop partition p2;  Query OK, 0 rows affected (0.01 sec)  Records: 0 Duplicates: 0 Warnings: 0  mysql> explain partitions select \* from test1 where submit\_time=date\_format('2010-02-15','%Y-%m-%d');  +------+-------------+-------+------------+------+---------------+------------+---------+-------+------+-------------+  | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | Extra |  +------+-------------+-------+------------+------+---------------+------------+---------+-------+------+-------------+  | 1 | SIMPLE | test1 | p3 | ref | time\_index | time\_index | 8 | const | 1 | Using where |  +------+-------------+-------+------------+------+---------------+------------+---------+-------+------+-------------+  **【规范】分区表的主键必须包含用于分区的列** |

## 分区表数据库分区键值支持的函数类型

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 要求内容 | 目前MYSQL支持的分区列的函数如下表,考虑使用分区表时必须参考下表.   |  |  |  | | --- | --- | --- | | [ABS()](http://dev.mysql.com/doc/refman/5.6/en/mathematical-functions.html#function_abs) | CEILING() (see CEILING() and FLOOR()) | [DAY()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_day) | | [DAYOFMONTH()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_dayofmonth) | [DAYOFWEEK()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_dayofweek) | [DAYOFYEAR()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_dayofyear) | | [DATEDIFF()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_datediff) | EXTRACT() (see EXTRACT() function with WEEK specifier) | FLOOR() (see CEILING() and FLOOR()) | | [HOUR()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_hour) | [MICROSECOND()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_microsecond) | [MINUTE()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_minute) | | [MOD()](http://dev.mysql.com/doc/refman/5.6/en/mathematical-functions.html#function_mod) | [MONTH()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_month) | [QUARTER()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_quarter) | | [SECOND()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_second) | [TIME\_TO\_SEC()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_time-to-sec) | [TO\_DAYS()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_to-days) | | [TO\_SECONDS()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_to-seconds) | UNIX\_TIMESTAMP() (permitted beginning with MySQL 5.6.1 and fully supported beginning with MySQL 5.6.3, withTIMESTAMP columns) | [WEEKDAY()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_weekday) | | [YEAR()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_year) |  | [YEARWEEK()](http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_yearweek) |   但是,其中只有YEAR(),TO\_DAYS(),TO\_SECONDS(),UNIX\_TIMESTEAMP()这类函数在进行优化器选择,其它函数除非where是采用等号匹配,否则不会选择使用优化器.  举例一:其它函数  mysql> create table sales(  -> money int unsigned not null,  -> date datetime  -> )  -> partition by range (year(date)\*100+month(date))(  -> partition q1 values less than (201204),  -> partition q2 values less than (201207),  -> partition q3 values less than (201210),  -> partition q4 values less than (201213)  -> );  Query OK, 0 rows affected (0.01 sec)  mysql> create index ix\_sales\_dt on sales(date);  Query OK, 0 rows affected (0.02 sec)  where是采用等号匹配,所以选择优化器  mysql> explain partitions select \* from sales where date=str\_to\_date('2012-01-01','%Y-%m-%d');  +------+-------------+-------+------------+------+---------------+-------------+---------+-------+------+-------------+  | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | Extra |  +------+-------------+-------+------------+------+---------------+-------------+---------+-------+------+-------------+  | 1 | SIMPLE | sales | q1 | ref | ix\_sales\_dt | ix\_sales\_dt | 9 | const | 1 | Using where |  +------+-------------+-------+------------+------+---------------+-------------+---------+-------+------+-------------+  1 row in set (0.00 sec)  where是采用范围匹配,所以没有选择优化器  mysql> explain partitions select \* from sales where date>=str\_to\_date('2012-02-01','%Y-%m-%d') and date<=str\_to\_date('2012-03-01','%Y-%m-%d');  +------+-------------+-------+-------------+-------+---------------+-------------+---------+------+------+-------------+  | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | Extra |  +------+-------------+-------+-------------+-------+---------------+-------------+---------+------+------+-------------+  | 1 | SIMPLE | sales | q1,q2,q3,q4 | range | ix\_sales\_dt | ix\_sales\_dt | 9 | NULL | 5 | Using where |  +------+-------------+-------+-------------+-------+---------------+-------------+---------+------+------+-------------+  1 row in set (0.00 sec)  举例二:推荐函数  mysql> create table sales(  -> money int unsigned not null,  -> date datetime  -> )  -> partition by range (year(date))(  -> partition q1 values less than (201204),  -> partition q2 values less than (201207),  -> partition q3 values less than (201210),  -> partition q4 values less than (201213)  -> );  Query OK, 0 rows affected (0.01 sec)  mysql> create index ix\_sales\_dt on sales(date);  Query OK, 0 rows affected (0.03 sec)  Records: 0 Duplicates: 0 Warnings: 0  where是采用范围匹配,选择优化器  mysql> explain partitions select \* from sales where date>=str\_to\_date('2012-02-01','%Y-%m-%d') and date<=str\_to\_date('2012-03-01','%Y-%m-%d');  +------+-------------+-------+------------+-------+---------------+-------------+---------+------+------+-------------+  | id | select\_type | table | partitions | type | possible\_keys | key | key\_len | ref | rows | Extra |  +------+-------------+-------+------------+-------+---------------+-------------+---------+------+------+-------------+  | 1 | SIMPLE | sales | q1 | range | ix\_sales\_dt | ix\_sales\_dt | 9 | NULL | 1 | Using where |  +------+-------------+-------+------------+-------+---------------+-------------+---------+------+------+-------------+  1 row in set (0.00 sec)  **【规范】YEAR(),TO\_DAYS(),TO\_SECONDS(),UNIX\_TIMESTEAMP()这类函数才可以用于分区表的分区里的函数处理.** |

## 数据严格检查与异常报错处理

|  |  |
| --- | --- |
| 要求内容 | **【规范】sql\_mode必须是STRICT\_TRANS\_TABLES模式**  set global sql\_mode="STRICT\_TRANS\_TABLES";  检查  mysql> select @@global.sql\_mode;  +---------------------+  | @@global.sql\_mode |  +---------------------+  | STRICT\_TRANS\_TABLES |  +---------------------+  1 row in set (0.00 sec)  mysql> |

## 数据严格检查与异常报错处理

|  |  |
| --- | --- |
| 要求内容 | MySQL管理数据是按页进行管理的，假定设定的页长度是16K，MySQL的真正记录就是存储到这个页中，简单理解就是记录在页内部按逻辑顺序做了排序存储，记录越短，当然一页能存储的记录数就越多,性能也是越好的.innodb内部对于每页最低要求存储2条记录，因此默认的存储格式compact，要求单条记录长度实际数据长度不能超过1982,如果超过1982怎么办呢?针对这个问题MySQL新设计Barracuda文件格式的dynamic和compressed存储格式，主要原理是在主键页面中列只存储前面的20个字节，多余的记录存储到另外的页面中(innodb中的专业术语是off-page).目前MYSQL默认是Barracuda文件格式的dynamic.建议建表的所有列的定义长度的总长度不要超过1982字节(如果是存汉字则需要除3),如果超过1982字节,性能会有开销.  一般发生在表中存在varchar,blob,text列的时候.  **【规范】每个页面至少写2行数据**  举例:  CREATE TABLE t\_item1(  id int(11) NOT NULL AUTO\_INCREMENT,  manufacture varchar(20) DEFAULT NULL,  name varchar(20) DEFAULT NULL,  PRIMARY KEY (id)  ) ENGINE=InnoDB DEFAULT CHARSET=UTF8 ROW\_FORMAT=DYNAMIC collate=utf8\_bin; |

## 索引包含的列的总长度限制

|  |  |
| --- | --- |
| 要求内容 | **【规范】对于varchar索引,字段总长度不能超过768个字节,**  建议组合索引的列的长度根据实际列数值的长度定义,比如身份证号定义长度为varchar(20),不要定位为varchar(100),同时,由于MYSQL默认采用UTF8字符集,一个字符3个字节,所以,实际索引所包含的列的长度要小于768/3=256字符长度.  例如:  mysql> CREATE TABLE t\_item1(  -> id int(11) NOT NULL AUTO\_INCREMENT,  -> manufacture1 varchar(128) DEFAULT NULL,  -> manufacture2 varchar(128) DEFAULT NULL,  -> manufacture3 varchar(128) DEFAULT NULL,  -> name varchar(20) DEFAULT NULL,  -> PRIMARY KEY (id)  -> ) ENGINE=InnoDB DEFAULT CHARSET=UTF8 ROW\_FORMAT=DYNAMIC collate=utf8\_bin;  Query OK, 0 rows affected (0.00 sec)  mysql> create index ind\_01 on t\_item1(manufacture1,manufacture2);  Query OK, 0 rows affected (0.00 sec)  Records: 0 Duplicates: 0 Warnings: 0  mysql> CREATE TABLE t\_item1(  -> id int(11) NOT NULL AUTO\_INCREMENT,  -> manufacture1 varchar(128) DEFAULT NULL,  -> manufacture2 varchar(128) DEFAULT NULL,  -> manufacture3 varchar(128) DEFAULT NULL,  -> name varchar(20) DEFAULT NULL,  -> PRIMARY KEY (id)  -> ) ENGINE=InnoDB DEFAULT CHARSET=UTF8 ROW\_FORMAT=DYNAMIC collate=utf8\_bin;  Query OK, 0 rows affected (0.00 sec)  mysql> create index ind\_01 on t\_item1(manufacture1,manufacture2,manufacture3);  ERROR 1071 (42000): Specified key was too long; max key length is 768 bytes |

## 建库必须增加区分大小写属性

|  |  |
| --- | --- |
| 要求内容 | **【规范】数据库collate属性为utf8\_bin.**  举例:  mysql> create database test2 default charset=utf8 collate=utf8\_bin;  Query OK, 1 row affected (0.00 sec) |

## Select for update注意事项（非自提交情况下）

|  |  |
| --- | --- |
| 要求内容 | **【规范】使用for update时SQL一定要走索引**  对于表products,有两列id和name,id是主键列,name是索引  例1:where条件明确指定主键,并且该行存在,则为row lock  SELECT \* FROM products WHERE id='3' FOR UPDATE;    例2:where条件不明确指定主键,则为通过索引列扫描到的数据结果都会lock  SELECT \* FROM products WHERE name='Mouse' FOR UPDATE;    例3:where条件不是索引列,id不是索引，不是主键,则为所有的记录lock  SELECT \* FROM products WHERE id<>'3' FOR UPDATE; |

## 库名,表名,字段,索引名必须统一使用小写字母

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,MySQL默认配置参数lower\_case\_table\_names为0,即库表名写存储和访问均大小写敏感,  2,如果大小写混合使用,可能存在abc,Abc,ABC等多个表共存,容易导致混乱,所以要求全部统一为小写.  3,字段名显式区分大小写,但实际使用不区分,即不可以建立两个名字一样但大小写不一样的字段,所以要求全部统一为小写.  4,为了统一规范, 库名、表名、字段名使用小写字母。  **库名、表名、字段名不要超过32个字符。**  库名、表名、字段名支持最多64个字符,但为了统一规范和性能开销的角度,建议不要超过32个字符。  **库名、表名、字段名禁止使用MySQL保留字。**  库名、表名、字段名等属性禁止使用MYSQL保留字,否则会导致报错。  举例一:表名大小写存在区别  **mysql> select count(\*) from fex.bapp\_auto\_task;** **+----------+** **| count(\*) |** **+----------+** **| 4 |** **+----------+** **1 row in set (0.00 sec)**  **mysql> select count(\*) from fex.BAPP\_AUTO\_TASK;** **ERROR 1146 (42S02): Table 'fex.BAPP\_AUTO\_TASK' doesn't exist**  举例二:控制表名只能小写  mysql> show global variables like "lower\_case\_table\_names"; +------------------------+-------+ | Variable\_name | Value | +------------------------+-------+ | lower\_case\_table\_names | 0 | +------------------------+-------+  mysql> create table TEST ( name char(20)); Query OK, 0 rows affected (0.00 sec) mysql> create table test ( name char(20));  Query OK, 0 rows affected (0.00 sec) mysql> insert into TEST set name = "a"; Query OK, 1 row affected (0.00 sec) mysql> insert into test set name = "b"; Query OK, 1 row affected (0.00 sec) mysql> select \* from TEST; +------+ | name | +------+ | a | +------+ 1 row in set (0.00 sec) mysql> select \* from test; +------+ | name | +------+ | b | +------+ 1 row in set (0.00 sec)  **修改参数为**lower\_case\_table\_names = 1后  mysql> show tables; +---------------------+ | Tables\_in\_case\_test | +---------------------+ | TEST | | test | +---------------------+ 2 rows in set (0.00 sec)  mysql> show global variables like "lower\_case\_table\_names"; +------------------------+-------+ | Variable\_name | Value | +------------------------+-------+ | lower\_case\_table\_names | 1 | +------------------------+-------+ 1 row in set (0.00 sec) MYSQL将大写的表转为小写  mysql> select \* from TEST; +------+ | name | +------+ | b | +------+ 1 row in set (0.00 sec) mysql> select \* from test; +------+ | name | +------+ | b | +------+ 1 row in set (0.00 sec)  MYSQL将大小写混合时候报错  mysql>create table TEst( name char(20));  ERROR 1050 (42S01): Table 'test' already exists mysql> insert into test set name = "c"; Query OK, 1 row affected (0.00 sec)  mysql> select \* from TEST; +------+ | name | +------+ | b | | c | +------+ 2 rows in set (0.00 sec)  mysql> select \* from test; +------+ | name | +------+ | b | | c | +------+ 2 rows in set (0.00 sec) |

## CHAR,VARCHAR的区别

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,char使用固定长度的空间进行存储，char(4)存储4个字符，如果需要存储的字符串的长度一样，适合用char。 2,varchar保存可变长度的字符串，varchar(10),除了需要存储10个字符，还需要1个字节存储长度信息(10),超过255的长度需要2个字节来存储。 3,char和varchar后面如果有空格，varchar会自动去掉空格后存储，char不会去掉空格，在进行字符串比较时，不会去掉空格进行比较。  4,数字类型尽量用数字类型存放不要用字符串保存，效率高，占用空间小。 |

## 区分使用DATETIME,TIMESTAMP,YEAR,DATE类型

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,DATETIME和TIMESTAMP都是精确到秒，优先选择TIMESTAMP，因为TIMESTAMP只有4个字节，而DATETIME需要8个字节。同时TIMESTAMP具有自动赋值以及自动更新的特性，格式为YYYY-MM-DD HH:MM:SS。  2,当一张表多有个  2,YEAR存放年的信息,格式为YYYY。  3,DATE存放日期的信息,格式为YYYY-MM-DD。  自动初始化,并自动更新:  column1 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP  只是自动初始化:  column1 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  自动更新,初始化的值为0:  column1 TIMESTAMP DEFAULT 0 ON UPDATE CURRENT\_TIMESTAMP  初始化的值为0:  column1 TIMESTAMP DEFAULT 0  举例一:  mysql> CREATE TABLE table1 (  -> ID INT unsigned NOT NULL AUTO\_INCREMENT,  -> T1 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -> T2 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -> T3 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -> Name varchar(50) DEFAULT NULL,  -> PRIMARY KEY (ID)  -> ) ;  Query OK, 0 rows affected (0.00 sec)  mysql> select \* from table1;  Empty set (0.00 sec)  mysql> insert into table1 (Name) values ('test');  Query OK, 1 row affected (0.00 sec)  mysql> select \* from table1\G;  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ID: 1  T1: 2014-12-04 15:45:44  T2: 2014-12-04 15:45:44  T3: 2014-12-04 15:45:44  Name: test  1 row in set (0.00 sec)  ERROR:  No query specified  mysql> update table1 set name='test111' where id=1;  Query OK, 1 row affected (0.00 sec)  Rows matched: 1 Changed: 1 Warnings: 0  mysql> select \* from table1\G;  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ID: 1  T1: 2014-12-04 15:45:44  T2: 2014-12-04 15:45:44  T3: 2014-12-04 15:45:44  Name: test111  1 row in set (0.00 sec)  ERROR:  No query specified  mysql> CREATE TABLE table1 (  -> ID INT unsigned NOT NULL AUTO\_INCREMENT,  -> T1 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  -> T2 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  -> T3 TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  -> Name varchar(50) DEFAULT NULL,  -> PRIMARY KEY (ID)  -> ) ;  Query OK, 0 rows affected (0.00 sec)  mysql> insert into table1 (Name) values ('test');  Query OK, 1 row affected (0.00 sec)  mysql> select \* from table1\G;  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ID: 1  T1: 2014-12-04 16:01:29  T2: 2014-12-04 16:01:29  T3: 2014-12-04 16:01:29  Name: test  1 row in set (0.00 sec)  ERROR:  No query specified  mysql> update table1 set name='test11' where id=1;  Query OK, 1 row affected (0.00 sec)  Rows matched: 1 Changed: 1 Warnings: 0  mysql> select \* from table1\G;  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  ID: 1  T1: 2014-12-04 16:01:47  T2: 2014-12-04 16:01:29  T3: 2014-12-04 16:01:47  Name: test11  1 row in set (0.00 sec)  ERROR:  No query specified |

## 不容许使用TEXT,BLOB类型

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,该类字段不易管理,并且访问的时候效率不高。  **2,将大字段,访问频率低的字段拆分到单独的表中存储,分离冷热数据,** 有利于有效利用缓存，防止读入无用的冷数据，较少磁盘IO，同时保证热数据常驻内存提高缓存命中率。 |

## SQL的WHERE条件的要求

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  Where条件必须用索引列作为列的过滤条件,如果列上使用函数处理,则函数处理尽量放在等号的右边。当有char或者varchar字段出现在where条件的左边时，右边的值必须使用引号引用，否则相当于在char或者varchar字段上使用了函数（to\_number），这将导致全表扫描。   1. Where条件同时用到多个索引列的时候,可以通过以下方法固定索引：   A,如果数据表里建立了很多索引,当MySQL对索引进行选择时,这些索引都在考虑的范围内,但有时我们希望MySQL只考虑几个索引,而不是全部的索引,这就需要用到USE INDEX对查询语句进行设置.  SELECT \* FROM TEST\_WH USE INDEX (INDEX\_NAME) …  B,如果数据表里建立了很多索引,而不被使用的索引又很少时,可以使用IGNORE INDEX进行反向选取.在上面的例子中是选择被考虑的索引,而使用IGNORE INDEX是选择不被考虑的索引.  SELECT \* FROM TEST\_WH IGNORE INDEX (INDEX\_NAME) …  C,如果数据表里建立了很多索引,我们希望MySQL必须要使用某一个索引(MySQL在查询时只能使用一个索引,因此只能强迫MySQL使用一个索引),这就需要使用FORCE INDEX来完成这个功能.  SELECT \* FROM TEST\_WH FORCE INDEX (INDEX\_NAME) … |

## **MYSQL中对NULL的处理**

|  |  |
| --- | --- |
| 要求内容 | mysql> select \* from test;  +------+------+  | a | b |  +------+------+  | 1 | NULL |  | 1 | test |  +------+------+  2 rows in set (0.00 sec)  mysql> select \* from test where b is null;  +------+------+  | a | b |  +------+------+  | 1 | NULL |  +------+------+  1 row in set (0.00 sec)  mysql> select \* from test where b=null;  Empty set (0.00 sec)  **【规范】**对于查询有null值列的数据必须使用is null查询 |

## 使用合理的分页方式以提高分页效率

|  |  |
| --- | --- |
| 要求内容 | 第一种分页写法：  select \* from t where thread\_id = 771025 and deleted =0 order by gmt\_create asc limit 0, 15;  原理：一次性根据过滤条件取出所有字段进行排序返回。  数据访问开销=索引IO+索引全部记录结果对应的表数据IO。  缺点：该种写法越翻到后面执行效率越差，时间越长，尤其表数据量很大的时候。  适用场景：当中间结果集很小（10000行以下）或者查询条件复杂（涉及多个不同查询字段或者多表连接）时适用。  第二种分页写法：  select t.\* from (select id from t where thread\_id = 771025 and deleted = 0 order by gmt\_create asc limit 0, 15) a, t where a.id = t.id;  前提：假设t表主键是id列，且有覆盖索引secondary key:(thread\_id, deleted, gmt\_create)  原理：先根据过滤条件利用覆盖索引取出主键id进行排序，再进行join操作取出其他字段。  数据访问开销=索引IO+索引分页后结果（例子中是15行）对应的表数据IO。  优点：每次翻页消耗的资源和时间都基本相同，就像翻第一页一样。  适用场景：当查询和排序字段（即where子句和order by子句涉及的字段）有对应覆盖索引时，且中间结果集很大的情况时适用。 |

## 拆分复杂SQL为多个小SQL,避免大事务

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,对于大的delete操作,要求使用delete+limit,将大事务拆分为小事务.  2,对于大的insert操作,要求使用将大的insert拆分为多个小事务.拆分规则:大于100万的数据量,单次提交小事务笔数最小为10万笔,小于100万的数据量, 单次提交小事务笔数最小为1万笔. |

## update跟着的子查询不能与update操作的是同一张表

|  |  |
| --- | --- |
| 要求内容 | **【规范】**update跟着的子查询不能与update操作的是同一张表  举例一:  update apples set price = (select price from apples where variety = 'gala') where variety = 'fuji';  会错误提示是：ERROR 1093 (HY000): You can't specify target table 'apples' for update in FROM clause. MySQL手册UPDATE documentation这下面有说明:Currently, you cannot update a table and select from the same table in a subquery. |

## 多表join的适用场景

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,对于分库分表的的场景,不容许有表的join是涉及到跨库的.  2,对于非分库分表的场景,join不能超过3张表. |

## SQL的group by,order by

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,对于分库分表的的场景,不容许使用 group by ,order by. |

## 自增长列的适用场景

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,对于分库分表的的场景,不容许使用自增列.  2.不建议将自增列作为业务主键.因为自增列不会持久化，有可能出现数据库重启后，获取的自增列的值比重启前已经获取过的值还小，这个bug预计在MySQL 8.0中修复。  举例:  重启前  mysql> create table t(id int auto\_increment primary key);  Query OK, 0 rows affected (0.28 sec)  mysql> insert into t values(),(),(),(),();  Query OK, 5 rows affected (0.11 sec)  Records: 5 Duplicates: 0 Warnings: 0  mysql> select \* from t;  +----+  | id |  +----+  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  +----+  5 rows in set (0.01 sec)  mysql> delete from t where id>=4;  Query OK, 2 rows affected (0.15 sec)  mysql> select \* from t;  +----+  | id |  +----+  | 1 |  | 2 |  | 3 |  +----+  3 rows in set (0.00 sec)  重启后  mysql> insert into t values();  Query OK, 1 row affected (0.23 sec)  mysql> select \* from t;  +----+  | id |  +----+  | 1 |  | 2 |  | 3 |  | 4 |  +----+  4 rows in set (0.07 sec)  如果自增列可以持久化的话，这里出现的值应该是6，而不是4. |

## 禁止使用存储过程、触发器、视图、自定义函数等

|  |  |
| --- | --- |
| 要求内容 | **【规范】**  1,不容许使用存储过程，触发器，视图，自定义函数。一旦数据库发生分库分表的动作后，之前在单库上运行的存储过程，触发器，视图，自定义函数均无法正常的调用或者逻辑不再和以前一样. |

## 命名规范

|  |  |
| --- | --- |
| 要求内容 | **【规范】**   1. 数据库对象必须使用小写字母，可与下划线数字组合。 2. 数据库对象命名禁止超过32个字符。 3. 数据库对象命名禁止使用MySQL保留字。 |

## 表设计规范

|  |  |
| --- | --- |
| 要求内容 | **【规范】**   1. 表必须有主键。 2. 主键字段不超过3个。 3. 表之间的关联查询使用主键作为关联字段。 4. 能使用整型字段作为主键，尽量使用整型字段。 5. 不能使用临时表，使用临时表会导致备份数据无法恢复。必须使用临时表业务需特别申请，无高可用保障。 |

## 字段类型设计规范

|  |  |
| --- | --- |
| 要求内容 | **【规范】**   1. 字段默认情况下尽可能为非空 NOT NULL 。 2. 字段默认情况尽可能设置默认值。 3. 越简单越好，将字符转化为数字、使用TINYINT代替ENUM类型。 4. 不使用负数值的字段须加入UNSIGNED属性。 5. 仅存储年使用YEAR类型。 6. 仅存储日期使用DATE类型。 7. 钱币等精确浮点类型使用DECIMAL类型。 8. 不允许使用ENUM。 9. 如果数值字段没有那么大，就不要用BIGINT。 |

## 索引设计规范

|  |  |
| --- | --- |
| 要求内容 | **【规范】**   1. 不使用更新频繁的列作为主键。 2. 不容许使用全文索引。 3. 索引创建选择唯一性较强的字段。 4. 索引选择数据类型较短的字段。 5. 合理创建联合索引，联合索引(a,b,c) 等于 (a) 、(a,b) 、(a,b,c)三个索引，一个索引中的字段数不超过5个，一张表不要超过4个索引。 6. 新建的唯一索引不能和主键重复。 7. 单张表的索引数量控制在5个以内。 8. 尽量避免使用外键，容易产生死锁，由上层应用程序保证约束。 9. 重要的SQL语句必须被索引。 10. Update\delete语句的where条件列。Update/delete一定要带where条件，并且where条件涉及到的列需建立索引。 11. order by \group by\distinct字段。 12. 多表JOIN的字段注意以下：  * 区分度最大的字段放在前面 * 核心SQL有限考虑覆盖索引 * 避免冗余和重复索引 * 索引要综合评估数据密度和分布以及考虑查询和更新比例 |

# 附录A：MySQL数据类型取值范围

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数值类型 | | | | |
| 类型 | 大小 | 范围（有符号） | 范围（无符号） | 用途 |
| TINYINT | 1 | (-128，127) | (0，255) | 小整数值 |
| SMALLINT | 2 | (-32 768，32 767) | (0，65 535) | 大整数值 |
| MEDIUMINT | 3 | (-8 388 608，8 388 607) | (0，16 777 215) | 大整数值 |
| INT或INTEGER | 4 | (-2 147 483 648，2 147 483 647) | (0，4 294 967 295) | 大整数值 |
| BIGINT | 8 | (-9 233 372 036 854 775 808，9 223 372 036 854 775 807) | (0，18 446 744 073 709 551 615) | 极大整数值 |
| FLOAT | 4 | (-3.402 823 466 E+38，1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38) | 0，(1.175 494 351 E-38，3.402 823 466 E+38) | 单精度  浮点数值 |
| DOUBLE | 8 | (1.797 693 134 862 315 7 E+308，2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308) | 双精度  浮点数值 |
| DECIMAL | 对DECIMAL(M,D) ，如果M>D，为M+2否则为D+2 | 依赖于M和D的值 | 依赖于M和D的值 | 小数值 |