# 增

## **1.非空约束，默认0**

ALTER TABLE `users` MODIFY COLUMN `rolesId` int(11) NOT NULL default 0

--必须增加 column 的type类型

## **2.外键约束**：

ALTER TABLE students ADD CONSTRAINT FK\_CLA\_ID FROEIGN KEY(cla\_id) REFERENCES class(cla\_id);

#外键中的级联关系有以下几种情况：

#ON DELETE CASCADE 删除主表中的数据时，从表中的数据随之删除

#ON UPDATE CASCADE 更新主表中的数据时，从表中的数据随之更新

#ON DELETE SET NULL 删除主表中的数据时，从表中的数据置为空

#默认 删除主表中的数据前需先删除从表中的数据，否则主表数据不会被删除

ALTER TABLE students ADD CONSTRAINT FK\_CLA\_ID FROEIGN KEY(cla\_id) REFERENCES class(cla\_id) ON DELETE CASCADE;

## **3.主键约束**：

--主键约束相当于(唯一约束+非空约束) 一张表中最多有一个主键约束

--删除主键约束前，如果有自增长需要先删除自增长,如果不删除自增长就无法删除主键约束

　 ALTER TABLE users MODIFY user\_id INT(10) PRIMARY KEY;

ALTER TABLE users CHANGE user\_id user\_id INT(10) PRIMARY KEY;

ALTER TABLE users ADD PRIMARY KEY(user\_id);

ALTER TABLE users ADD CONSTRAINT PK\_ID PRIMARY KEY(user\_id);

## **4.唯一约束**：

--如果已经有重复值，会报错，无法增加唯一但是可以为空(空和空不相等)

ALTER TABLE users CHANGE username username VARCHAR(20) UNIQUE;

ALTER TABLE users ADD UNIQUE(username);

## **5.检查约束**：

-- mysql不支持检查约束，但是写上检查约束不会报错

　　ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK (char\_length(zipcode) = 5);

## 6.在新增数据的时候给一个增加时间

ALTER TABLE `sys\_importance`

ADD COLUMN `creatTime` timestamp NULL DEFAULT CURRENT\_TIMESTAMP COMMENT '数据创建时间' AFTER `latitude`;

## 7.数据自增长

ALTER TABLE t\_user MODIFY user\_id INT(10) AUTO\_INCREMENT;

ALTER TABLE t\_user CHANGE user\_id user\_id INT(10) AUTO\_INCREMENT;

# 删

## **0.删除一个字段**：

　　ALTER TABLE distributors DROP COLUMN address RESTRICT;

## **1.非空约束**：

　　ALTER TABLE distributors ALTER COLUMN street DROP NOT NULL;

## 2.外键约束

ALTER TABLE students DROP FOREIGN KEY FK\_CLA\_ID;

## 3.主键约束

ALTER TABLE t\_user DROP PRIMARY KEY;

## 4.唯一约束

ALTER TABLE t\_user DROP INDEX user\_id;

## **5.删除一个表和它的所有子表的检查约束**：

　　 ALTER TABLE distributors DROP CONSTRAINT zipchk;

## 6.删除自增长

ALTER TABLE t\_user MODIFY user\_id INT(10);

ALTER TABLE t\_user CHANGE user\_id user\_id INT(10);

# 改

## 结构

### 1.向表中增加一个 varchar 列：

ALTER TABLE S\_ADVISE ADD DISAGREE\_REASON varchar(255) NULL

### 2.修改两个现有字段的类型：

　　ALTER TABLE distributors ALTER COLUMN address TYPE varchar(80),　　ALTER COLUMN name TYPE varchar(100);

### 3.对现存字段改名：

ALTER TABLE `users`

CHANGE COLUMN `password` `password11` varchar(20) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT 123456;

## 值

### 1.字段值

update table\_name set col\_value1=xxxx where col\_value2='xxx';

### 2.把表移动到另外一个表空间：

　　ALTER TABLE distributors SET TABLESPACE fasttablespace;

# 查

## 1.字母开头

select \* from s\_user where account like '[a-z]%';

## 2.分页

select \* from (select row\_number()over(order by id)rownumber,\* from S\_USER)a where rownumber>30 and rownumber<40;

## 3.从查询information\_schema中查询指定表中的约束

USE INFORMATION\_SCHEMA;

SELECT CONSTRAINT\_NAME FROM TABLE\_CONSTRAINTS WHERE TABLE\_NAME='student';

使用一个 USING 子句， 把一个包含 UNIX 时间戳的 integer 字段转化成 timestamp with time zone：

　　ALTER TABLE foo

　　ALTER COLUMN foo\_timestamp TYPE timestamp with time zone

　　USING

timestamp with time zone 'epoch' + foo\_timestamp \* interval '1 second';

# 优化

## 优化点

1、连接数据库的问题：建立连接和关闭连接的次数太多，导致IO访问次数太频繁。

 2、应该使用批量插入和批量修改的方法，而不是有一条数据就进行插入，这样会导致访问数据库的实际特别的慢。

 3、在建立库的时候要建立适当的索引：如主键、外键、唯一等，优化查询效率。

## 查询语句优化

1、应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

2、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：   
     select id from t where num is null   
     可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：   
     select id from t where num=0

3、尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：   
     select id from t where num=10 or num=20   
     可以这样查询：   
     select id from t where num=10   
     union all   
     select id from t where num=20

4、下面的查询也将导致全表扫描：

    select id from t where name like ‘%abc%’

    若要提高效率，可以考虑全文检索。

5、in 和 not in 也要慎用，否则会导致全表扫描，如：   
     select id from t where num in(1,2,3)   
     对于连续的数值，能用 between 就不要用 in 了：   
     select id from t where num between 1 and 3

6、如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：   
     select id from t where num=@num   
     可以改为强制查询使用索引：   
     select id from t with(index(索引名)) where num=@num

7、应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：   
     select id from t where num/2=100   
     应改为:   
     select id from t where num=100\*2

8、应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：   
     select id from t where substring(name,1,3)=’abc’–name以abc开头的id   
     select id from t where datediff(day,createdate,’2005-11-30′)=0–’2005-11-30′生成的id   
     应改为:   
     select id from t where name like ‘abc%’   
     select id from t where createdate>=’2005-11-30′ and createdate<’2005-12-1′

9、不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

10、在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

11、不要写一些没有意义的查询，如需要生成一个空表结构：   
     select col1,col2 into #t from t where 1=0   
     这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：   
     create table #t(…)

12、很多时候用 exists 代替 in 是一个好的选择：   
     select num from a where num in(select num from b)   
     用下面的语句替换：   
     select num from a where exists(select 1 from b where num=a.num)

## ****其他需要注意的地方：****

1、尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

2、任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。

3、尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

4、避免频繁创建和删除临时表，以减少系统表资源的消耗。

5、临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

6、在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

7、如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

8、尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

9、使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

10、与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

11、在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。

12、尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

13、尽量避免大事务操作，提高系统并发能力。

# NoSQL 数据库分类

|  |  |  |
| --- | --- | --- |
| 类型 | 部分代表 | 特点 |
| 列存储 | Hbase  Cassandra  Hypertable | 顾名思义，是按列存储数据的。最大的特点是方便存储结构化和半结构化数据，方便做数据压缩，对针对某一列或者某几列的查询有非常大的IO优势。 |
| 文档存储 | MongoDB  CouchDB | 文档存储一般用类似json的格式存储，存储的内容是文档型的。这样也就有有机会对某些字段建立索引，实现关系数据库的某些功能。 |
| key-value存储 | Tokyo Cabinet / Tyrant  Berkeley DB  MemcacheDB  Redis | 可以通过key快速查询到其value。一般来说，存储不管value的格式，照单全收。（Redis包含了其他功能） |
| 图存储 | Neo4J  FlockDB | 图形关系的最佳存储。使用传统关系数据库来解决的话性能低下，而且设计使用不方便。 |
| 对象存储 | db4o  Versant | 通过类似面向对象语言的语法操作数据库，通过对象的方式存取数据。 |
| xml数据库 | Berkeley DB XML  BaseX | 高效的存储XML数据，并支持XML的内部查询语法，比如XQuery,Xpath。 |