# TAOBAO-数据库开发使用文档-v1

**--淘宝 DBA 团队**

一. 数据库设计

1. 字段类型规范

在 oracle 中，我们常用的数据类型为 number,varchar2,date 三种，对于 number 类型，在表结构设计时，不用指定精度，这个精度在应用层自己控制；对于 varchar2 字符串类型，在数据库端也要定义具体的长度，比如说 nick varchar2(32),在应用端需要作长度校验；对于 date 类型，因没有长度的概念，没有任何要求:

示例：

```
SQL> create table test(id number not null,
  2   nick varchar2(32) not null,
  3   gmt_create date not null,
  4   gmt_modified date not null);
```

在 mysql 中，我们常用的数字类型有 tinyint,int , bigint ，对于自动增长的数字类型的主键 auto_increment, 都要求是是无符号型的，示例：id unsigned bigint auto_increment not null ,并根据此表未来的数据量评估，如果未来会上亿，则需要用 bigint 类型，以避免溢出;如果表的数据量较小则用 int 足够，对于一些状态字段取值，则建议使用 tinyint;字符类型 varchar,需要指明长度 ; 时间类型为 datetime

示例：

```
mysql> create table test(id bigint unsigned auto_increment not null,
nick varchar(32) not null,
gmt_create datetime not null,
gmt_modified datetime not null
primary key(id));
```

在进行表数据类型选择的时候，不建议使用大字段：blob，clob，text；

定义字符类型的时候，要严格控制字符串的长度，不能随意指定字符的长度，需要严格定义最大需求长度，如果可以的话前端应用控制字符的输入数量；

NULL 在数据库中有着特殊的含义，从业务上，最好不要使用在业务场景中，往往后期在处理 null 值数据的时候,给应用带来足多麻烦;从数据库上,null 给数据库增加负担，索引的使用效率也不好（oracle 中 null 值不能使用索引）；数字类型的可以指定一个特殊值，如负数；字符型的可以指定空字符串，或其他特殊字符；日期类型的可以使用 0000-00-00 00:00:00;

2. 命名规范:

表名或列名要以字符开头，不能以数字开头,不能使用 oracle or mysql 数据库中的关键字,最好以业务线来命名表名，比如无名良品的产品线的都要 ali_开头，开发平台的以 top_开头;有特殊用途的表如临时表需要以 tmp_开头,然后加上该临时表的责任人如：tmp_xuancan_；

表名或列名要以下划线分割，如 bmw_users 表，user_id 用户数字 ID；并且在命名上要保持统一，如在表 a,表 b 中都有用户数字 ID，表 a 中命名为 user_id,而表 b 中命名为 userid,这是不允许的；

表名的长度不要超过 32 个字符，表，列都采用小写；

3．Sql 翻页写法：

数据库翻页的原理：一条 SQL 计算总数量 count(*)→一条 SQL 返回分页后的数据
oracle 数据库一般采用 rownum 来进行分页，常用分页语法有如下两种：
直接通过 rownum 分页：

```
select * from (
        select a.*,rownum rn from
                (select * from product a where company_id=? order by status) a
        where rownum<=20)
where rn>10;
```

数据访问开销=索引 IO+索引全部记录结果对应的表数据 IO；
采用 rowid 分页：
优化原理是通过纯索引找出分页记录的 ROWID，再通过 ROWID 回表返回数据，要求内层查询和排序字段全在索引里。

```
create index myindex on product(company_id,status);
 select b.* from (
        select * from (
                select a.*,rownum rn from
                        (select rowid rid,status from product a where company_id=? order
by status) a
                where rownum<=20)
        where rn>10) a, product b
where a.rid=b.rowid;
```

数据访问开销=索引 IO+索引分页结果对应的表数据 IO
实例：
一个公司产品有 1000 条记录，要分页取其中 20 个产品，假设访问公司索引需要 50 个 IO，2 条记录需要 1 个表数据 IO。
那么按第一种 ROWNUM 分页写法，需要 550(50+1000/2)个 IO，按第二种 ROWID 分页写法，只需要 60 个 IO(50+20/2);

Mysql 翻页写法：

优化写法：(先根据过滤条件取出主键id进行排序，再进行join操作取出其他相关字段）
示例：

```
select  t.id, t.thread_id, t.group_id, t.deleted,
t.author_id,t.author_nick, t.author_ip,
t.content, t.last_modified, t.gmt_create, t.gmt_modified, t.floor,
t.reply, t.reply_time from
(select id from reply_0029 where thread_id = 771025 and deleted = 0 order
by gmt_create asc limit 0, 15) a, reply_0029 t where a.id = t.id;
```

普通写法 ：(一次性根据过滤条件取出所有字段进行排序返回）示例：

```
select  t.id, t.thread_id, t.group_id, t.deleted, t.author_id,
t.author_nick, t.author_ip,
t.content, t.last_modified, t.gmt_create, t.gmt_modified, t.floor,
t.reply, t.reply_time from reply_0029 t
where thread_id = 771025 and deleted = 0 order by gmt_create asc limit
0, 15;
```

二．数据订正：

线上所有数据变更，不论是项目，日常，临时数据订正都需要走数据订正流程，对于我们常见的项目或日常发布时，数据初始化也是要走这个流程的；在订正的 sql 脚本中需要加上 gmt_modified 字段；

大数据量订正的时候，在存储过程中一定要加入批量提交的功能，如 100 条提交一次；订正实现过程中，最好采用中间表的方式来订正；mysql 存储过程订正的过程中主要 autocommit 关闭；

```
--创建临时表
create table tmp_xf_id_del as
select id from xf_user_info where status=2 and gmt_create< '2010-07-01';

delimiter // --使用//带代替;，这个符号可以自己定义
delimiter ; --换回;

set autocommit =0; --设置为非自动提交，脚本中定义为 100 条提交一次
```

---

```
--创建存储过程
CREATE PROCEDURE sp_xf_del_test()
begin
```

```
    declare v_exit int default 0;
    declare v_id bigint;
    declare i int default 0;
    declare c_ids cursor for select id from tmp_xf_id_del;
    declare  continue handler for not found set v_exit=1;
    open c_ids;
    repeat
        fetch c_ids into v_id;
          delete from xf_user_info where id  = v_id;
          set i=i+1;
          if mod(i,100)=0 then commit;
        end if;
    until v_exit=1
    end repeat;
    close c_ids;
    commit;
end;
//
```

---

--调用存储过程

call sp_xf_del_test()//

Oracle 的数据订正存储过程示例：

```
Declare
   Num number :=0;
Begin
   For cur1 in (select rowid rid from table_name t where need_modify is null) loop
        Update table_name t set t.column_name=1，gmt_modified=sysdate where
rowid=cur1.rid;
        Num :=num +1;
   If mod(num,1000) =0 then
       Commit;
   End if;
   End loop;
Commit;
exception
   when others then
```

```
    v_errmsg := sqlerrm;
    insert into tmp_debug(debug_id, error_msg, error_time) values('xc_sjdz_1230', v_errmsg,
sysdate);
    commit;
end;
/
```

三．DDL 变更申请：

现在线上的所有 DDL 变更都是通过 websql 中的建表工具：创建新表，修改表结构；在提生产变更的时候需要指定该表上线的时间，在项目日常预发前一天会通知到开发和 DBA，并安排晚上值周的同事执行；对于特殊情况下的 DDL 需要临时执行，需向主管申请；

四．Sql 审核：

Sql 审核是在开发工程师将代码提交测试之前，交由 DBA 审核，审核通过后，DBA 会在日常测试环境中对表建立索引；

Sql 提交方式：review board/word 文档；

Review board 需要安装相关插件，sql map 文件中需要将相关信息填写清楚；

word 文档，由统一的模板填写，开发工程师将本次项目日常中改动，新增的 sql 填写到模板中；

在 sqlmap 或者 word 文档中必须注明 sql 的执行频率，功能，部署在前后台，返回的结果集大小，不按模板填写的，不以通过；

语句 1：新增/变更已有
功能：
预计每天该 SQL 语句的调用频率（单位：次/小时）：
布署在前台还是后台：前台/后台
负责人：
结果集大小：

```
<sqlmap……>
……
</sqlmap>
```

六．常用链接：

流程：

1.DBA 数据订正流程

2.DBA 内部审核流程

3.DBA 数据库上线服务器申请

4.连接池申请变更流程

[5.Sql 审核流程]

七．数据库工具：

**websql**：http://dba.tools.taobao.com:9999/workspace.htm

建表工具：新建表，对已有表修改，关键字过滤，可生成 excel 表格；

【自动生成 javaBean；hibernate 注解说明；iBATIS 配置文件:update,insert,delete；常规的 javadao 层操作模板代码】--该功能在 4 月中旬发布，将会极大减小开发同学的数据库编码工作^_^

查询：查表，查数据库，查少量数据（分表，分库，线上数据查询，导出）

**北斗系统：appops/123456**

http://beidou.taobao.net/index.php?controller=users&action=login

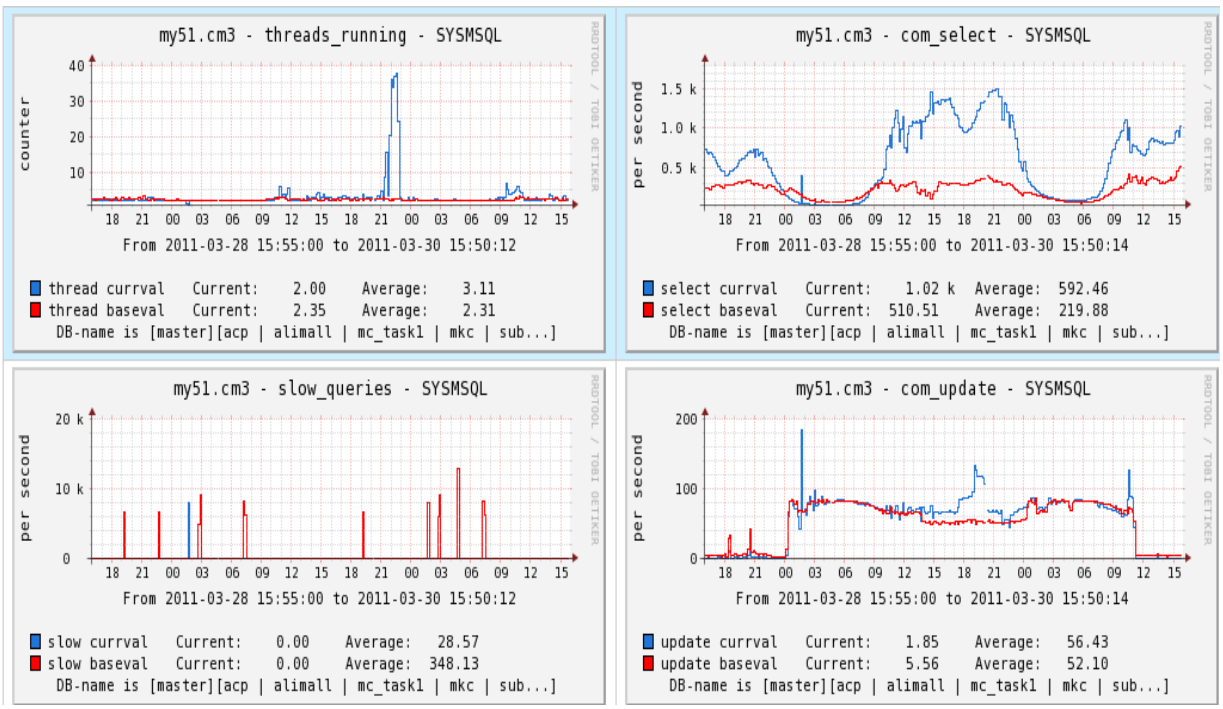

Mysql awr： 展示当前数据库 sql 执行情况


| sqlid | sql_fulltext | 详情 | new | exe | ex% | gf% | ela | el% | 操作 | 历史 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1801837593 | select * from acp_vote where id = :d; | 查看 | NO | 0.57M | 43.66 | 162.57 | 0.15 | 5.02 | 拷贝 | 查看 |
| 3859986604 | select * from acp_vote_result where vote_id = :d order by win_num desc; | 查看 | NO | 0.56M | 43.31 | 181.70 | 0.41 | 13.43 | 拷贝 | 查看 |
| 1372878486 | prepare select * from ap_attract_invest where auditor = ?; | 查看 | NO | 37.29K | 2.79 | 800.00 | 0.06 | 0.14 | 拷贝 | 查看 |
| 2507708893 | execute select * from ap_attract_invest where auditor = :d; | 查看 | NO | 35.66K | 2.67 | 800.00 | 35.63 | 72.55 | 拷贝 | 查看 |
| 2991518513 | set autocommit=:d; | 查看 | NO | 12.62K | 0.95 | -71.62 | 0.09 | 0.06 | 拷贝 | 查看 |
| 2516098119 | select * from message_task:d where task_status = :d and executor = :d an | 查看 | NO | 9.27K | 0.69 | -64.20 | 0.42 | 0.22 | 拷贝 | 查看 |
| 3437743687 | prepare select * from ali_mall_user where id=?; | 查看 | NO | 8.75K | 0.66 | 701.88 | 0.15 | 0.07 | 拷贝 | 查看 |
| 3612272835 | /* mysql-connector-java:d:d ( revision: ${svn.revision} ) */select @@ses | 查看 | NO | 7.1K | 0.53 | 800.00 | 2.15 | 0.87 | 拷贝 | 查看 |
| 166949269 | select @@session.tx_isolation; | 查看 | NO | 6.75K | 0.51 | 800.00 | 1.68 | 0.65 | 拷贝 | 查看 |
| 3678807309 | prepare select * from ali_selection_commodity com, ali_selection_commodi | 查看 | YES | 6.62K | 0.5 | 0.00 | 0.19 | 0.07 | 拷贝 | 查看 |
| 3690769690 | execute select * from ali_selection_commodity com, ali_selection_commodi | 查看 | YES | 6.57K | 0.49 | 0.00 | 0.24 | 0.09 | 拷贝 | 查看 |
| 292153665 | execute select * from ali_mall_user where id=:d; | 查看 | NO | 6.21K | 0.47 | 610.85 | 0.2 | 0.07 | 拷贝 | 查看 |
| 2650070603 | prepare select * from ali_point_acc where id =?; | 查看 | NO | 2.73K | 0.2 | -57.13 | 0.15 | 0.02 | 拷贝 | 查看 |
| 42421084 | select * from acp.acp_leave_message where board_id = :d and status in ( | 查看 | NO | 2.7K | 0.2 | -93.18 | 7.39 | 1.14 | 拷贝 | 查看 |

Oracle awr：


数据库的 awr 对于开发人员了解数据库执行 sql 情况最理想的工具，项目日常刚刚发布上去，或者运行一段时间后，开发人员可以通过数据库 awr 报表，对 sql 的执行情况了然于胸；

主机监控 ： 主机 tps/cpu/memory/threads/network



主机监控可以展现当前数据库服务器的 load/tps/cpu/memory/threads/network 等指标图像化的展示，可以很好的清楚判断业务的压力，为应用，数据库的架构方案给予充分的依据支持。

七．Sql 编写建议：

**一优化数据访问：**

**1.1.只返回需要的字段**

通过去除不必要的返回字段可以提高性能，例：

调整前：select * from product where company_id=?;

调整后：select id,name from product where company_id=?;

优点：

1、减少数据在网络上传输开销

2、减少服务器数据处理开销

3、减少客户端内存占用

4、字段变更时提前发现问题，减少程序 BUG

5、如果访问的所有字段刚好在一个索引里面，则可以使用纯索引访问提高性能。

缺点：增加编码工作量

由于会增加一些编码工作量，所以一般需求通过开发规范来要求程序员这么做，否则等项目上线后再整改工作量更大。

如果你的查询表中有大字段或内容较多的字段，如备注信息、文件内容等等，那在查询表时一定要注意这方面的问题，否则可能会带来严重的性能问题。如果表经常要查询并且请求大内容字段的概率很低，我们可以采用分表处理，将一个大表分拆成两个一对一的关系表，将不常用的大内容字段放在一张单独的表中。如一张存储上传文件的表：

T_FILE（ID,FILE_NAME,FILE_SIZE,FILE_TYPE,FILE_CONTENT）

我们可以分拆成两张一对一的关系表：

T_FILE（ID,FILE_NAME,FILE_SIZE,FILE_TYPE）

T_FILECONTENT（ID, FILE_CONTENT）

通过这种分拆，可以大大提少 T_FILE 表的单条记录及总大小，这样在查询 T_FILE 时性能会更好，当需要查询 FILE_CONTENT 字段内容时再访问 T_FILECONTENT 表。

2 重构查询：

2.1 缩短查询:这种方法在其本质上是一种分而治之的办法，把一个大事务分解成多个小事务，这样就可以减少影响的行数，缩短锁表时间，同时减少对事务日志，回滚日志的生成；典型的应用场景为周期性的清理数据任务：

```
Delete from task where gmt_create<date_sub(now(),interval 3 month);
```

对应的程序代码可以改为：

```
Rows_affects=0
Do{
Rows_affects=do_query(
    “delete from task where gmt_create<date_sub(now(),interval 3 month) limit
    1000”)
}while row_affects>0
```

3 分解连接：

许多高性能的网站都用了分解连接技术，可以把一个多表连接分解成多个单个查询，也就是在应用端做了数据库连接功能，在典型的 OLTP 交易系统中尤其重要：

```
Select * from tag join tag_post on tag_post.tag_id=tag.id
                    Join post on tag_post.post_id=post.id
Where tag.tag='mysql';
```

可以用下面的语句来代替：

→Select * from tag where tag='mysql';

→Select * from tag_post where tag_id=1234;

→Select * from post where post.id in(123,456,789);

小结：什么时候应用查询端进行连接效率更高：

(1)可以缓存最开始查询中的大量数据(tag 表中的数据更新变动不大，所以程序可以缓存该表数据于内存中，这样第一个查询就可以跳过；

(2)使用 myisam 表，由于组合查询的时候，单表锁表时间为该查询的锁表时间，分开后的单个查询对表锁时间要求降低；

(3)数据分布在不同的数据库服务器上；

(4)对于大表使用 in()替换联接，在第三个查询中，我们传入的 in list 为排好顺序的结果集，这样能更高效的读取数据；如果为联接查询，则 mysql 需要对他们进行排序；

(5)一个连接引用了一个表多次；

4 是否将将查询结果缓存到内存：

绝大多数连接 mysql 的类库能让你提取完整的结果然后缓存到内存中，默认的行为通常是提取所有的数据，然后缓存，mysql 只有在所有的数据提取之后，才会释放掉所有的锁和资源，查询的状态为 sending data；

通常情况下，这种将数据缓存在类库内存中的处理方式会工作良好，但是在处理庞大的结果集的时候也许会需要很长的时间和大量的内存，这个时候就需要不用缓存数据了，这样做的缺点在于应用查询和类库交互的时候，需要服务器端的锁和资源都是被锁定的。

Php：mysql_unbuffered_query

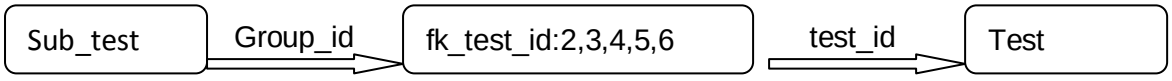
Perl：prepare()

5 子查询限制：

在 mysql 中对子查询优化的有一些不尽人意：

```
Select * from test where test_id in(select fk_test_id from sub_test where group_id=10)
```

通常我们会想到该 sql 的执行顺序为：




sub_test 表中根据 group_id 取得 fk_test_id(2,3,4,5,6),然后在到 test 中，带入 test_id=2,3,4,5,6 取得查询数据，但是实际 mysql 的处理方式为：

```
Select * from test where exists (
select * from sub_test where group_id=10 and sub_test.test_id=test.id)
```

mysql 将会扫描 test 中的所有数据，每条数据将会传到子查询中与 sub_test 关联，子查询不能首先被执行，如果 test 表很大的话，那么性能上将会出现问题，这时候就需要改写查询了：

```
select t1.* from test t1,sub_test t2 where t1.test_id=t2.fk_test_id and t2.group_id=10;
```

6 使用 exists 代替 distinct；

Exists 在 sql 语句中代表‘有一个匹配’的概念,他不会产生任何重复的行，那么就能够代替 group by 和 distinct[这两个操作没有索引可以使用的情况下，需要对查询结果集排序]：

```
Select distinct test.test_id from test , sub_test where sub_test.test_id=test.id
```

```
Select test_id from test where exists (
select * from sub_test where sub_test.test_id=test.id)
```

7 使用 union all 代替 or：

通常有下面的一种情况：

```
Select test_id,test_group from test where fk_test_id=10 or group_id=28;
```

表中单独为这两个查询条件建立有两个索引，这种情况可以即使优化器可以使用索引合并来完成查询，但是有时候这种算法需要花费大量的 cpu 和内存资源，那么就需要对 sql 需要改写了：

```
Select test_id,test_group from test where fk_test_id=10
Union all
Select test_id,test_group from test where group_id=28 and fk_test_id<>10;
```

**二．优化查询条件**

SQL 什么条件会使用索引?

当字段上建有索引时，通常以下情况会使用索引：

INDEX_COLUMN = ?
INDEX_COLUMN > ?
INDEX_COLUMN >= ?
INDEX_COLUMN < ?
INDEX_COLUMN <= ?
INDEX_COLUMN between ? and ?
INDEX_COLUMN in (?,?,...,?)
INDEX_COLUMN like ?||'%'（后导模糊查询）
T1. INDEX_COLUMN=T2. COLUMN1（两个表通过索引字段关联）

SQL 什么条件不会使用索引?

| 查询条件 | 不能使用索引原因 |
|---|---|
| INDEX_COLUMN <> ?<br>INDEX_COLUMN not in (?,?,...,?) | 不等于操作不能使用索引 |
| function(INDEX_COLUMN) = ?<br>INDEX_COLUMN + 1 = ?<br>INDEX_COLUMN \|\| 'a' = ? | 经过普通运算或函数运算后的索引字段不能使用索引 |
| INDEX_COLUMN like '%'\|\|?<br>INDEX_COLUMN like '%'\|\|?\|\|'%' | 含前导模糊查询的Like语法不能使用索引 |
| INDEX_COLUMN is null | B-TREE 索引里不保存字段为 NULL 值记录，因此 IS NULL 不能使用索引 |

| | |
|---|---|
| NUMBER_INDEX_COLUMN='12345'<br>CHAR_INDEX_COLUMN=12345 | Oracle在做数值比较时需要将两边的数据转换成同一种数据类型，如果两边数据类型不同时会对字段值隐式转换，相当于加了一层函数处理，所以不能使用索引。 |
| a.INDEX_COLUMN=a.COLUMN_1 | 给索引查询的值应是已知数据，不能是未知字段值。 |

注：

经过函数运算字段的字段要使用可以使用函数索引，这种需求建议与 DBA 沟通。

有时候我们会使用多个字段的组合索引，如果查询条件中第一个字段不能使用索引，那整个查询也不能使用索引

如：我们 company 表建了一个 id+name 的组合索引，以下 SQL 是不能使用索引的

Select * from company where name=?

Oracle9i 后引入了一种 index skip scan 的索引方式来解决类似的问题，但是通过 index skip scan 提高性能的条件比较特殊，使用不好反而性能会更差。

我们一般在什么字段上建索引？

这是一个非常复杂的话题，需要对业务及数据充分分析后再能得出结果。主键及外键通常都要有索引，其它需要建索引的字段应满足以下条件：

1、字段出现在查询条件中，并且查询条件可以使用索引；

2、语句执行频率高，一天会有几千次以上；

3、通过字段条件可筛选的记录集很小，那数据筛选比例是多少才适合？

这个没有固定值，需要根据表数据量来评估，以下是经验公式，可用于快速评估：

小表(记录数小于 10000 行的表)：筛选比例<10%；

大表：(筛选返回记录数)<(表总记录数*单条记录长度)/10000/16

单条记录长度≈字段平均内容长度之和+字段数*2

以下是一些字段是否需要建 B-TREE 索引的经验分类：

| | 字段类型 | 常见字段名 |
|---|---|---|
| 需要建索引的字段 | 主键 | ID,PK |
| | 外键 | PRODUCT_ID,COMPANY_ID,MEMBER_ID,ORDER_ID,TRADE_ID,PAY_ID |
| | 有对像或身份标 | HASH_CODE,USERNAME,IDCARD_NO,EMAIL,TEL_NO,IM_NO |

| | | |
|---|---|---|
| | 识意义字段 | |
| 索引慎用字段，需要进行数据分布及使用场景详细评估 | 日期 | GMT_CREATE,GMT_MODIFIED |
| | 年月 | YEAR,MONTH |
| | 状态标志 | PRODUCT_STATUS,ORDER_STATUS,IS_DELETE,VIP_FLAG |
| | 类型 | ORDER_TYPE,IMAGE_TYPE,GENDER,CURRENCY_TYPE |
| | 区域 | COUNTRY,PROVINCE,CITY |
| | 操作人员 | CREATOR,AUDITOR |
| | 数值 | LEVEL,AMOUNT,SCORE |
| | 长字符 | ADDRESS,COMPANY_NAME,SUMMARY,SUBJECT |
| 不适合建索引的字段 | 描述备注 | DESCRIPTION,REMARK,MEMO,DETAIL |
| | 大字段 | FILE_CONTENT,EMAIL_CONTENT |

如何知道 SQL 是否使用了正确的索引?

简单 SQL 可以根据索引使用语法规则判断，复杂的 SQL 不好办，判断 SQL 的响应时间是一种策略，但是这会受到数据量、主机负载及缓存等因素的影响，有时数据全在缓存里，可能全表访问的时间比索引访问时间还少。要准确知道索引是否正确使用，需要到数据库中查看 SQL 真实的执行计划，这个话题比较复杂，详见 SQL 执行计划专题介绍。

索引对 DML(INSERT,UPDATE,DELETE)附加的开销有多少？

这个没有固定的比例，与每个表记录的大小及索引字段大小密切相关，以下是一个普通表测试数据，仅供参考：

索引对于 Insert 性能降低 56%

索引对于 Update 性能降低 47%

索引对于 Delete 性能降低 29%

因此对于写 IO 压力比较大的系统，表的索引需要仔细评估必要性，另外索引也会占用一定的存储空间。

3.fetch size

当我们采用 select 从数据库查询数据时，数据默认并不是一条一条返回给客户端的，也不是一次全部返回客户端的，而是根据客户端 fetch_size 参数处理，每次只返回 fetch_size 条记录，当客户端游标遍历到尾部时再从服务端取数据，直到最后全部传送完成。所以如果我们要从服务端一次取大量数据时，可以加大 fetch_size，这样可以减少结果数据传输的交互次数及服务器数据准备时间，提高性能。

以下是 jdbc 测试的代码，采用本地数据库，表缓存在数据库 CACHE 中，因此没有网络连接及磁盘 IO 开销，客户端只遍历游标，不做任何处理，这样更能体现 fetch 参数的影响：

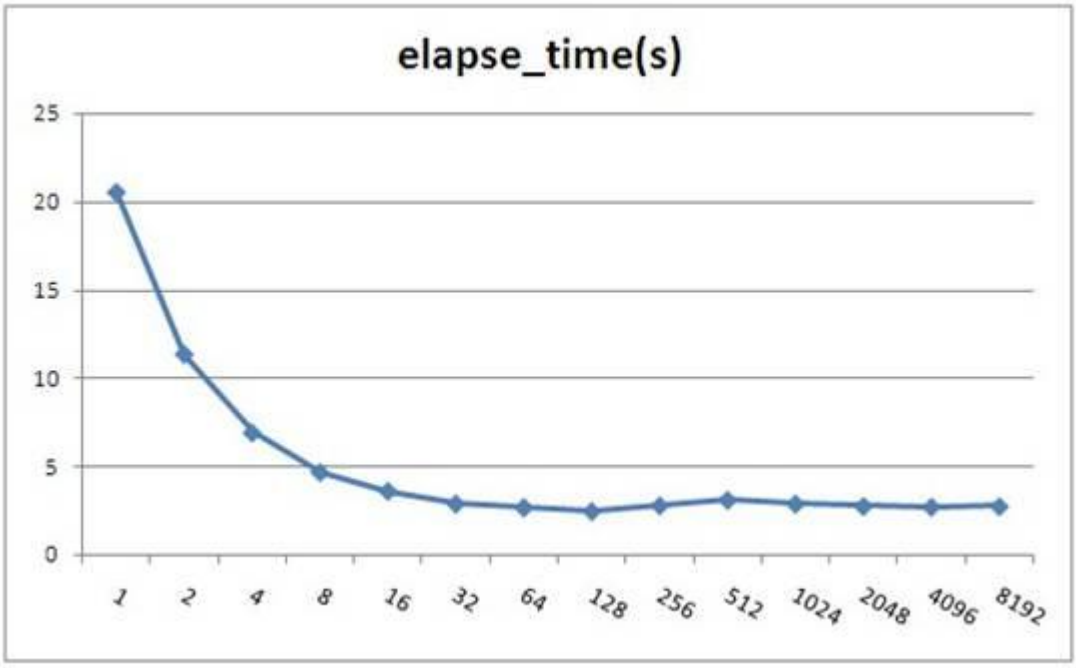
```
String vsql ="select * from t_employee";
PreparedStatement pstmt =
conn.prepareStatement(vsql,ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_ONLY);
pstmt.setFetchSize(1000);
ResultSet rs = pstmt.executeQuery(vsql);
int cnt = rs.getMetaData().getColumnCount();
Object o;
while (rs.next()) {
   for (int i = 1; i <= cnt; i++) {
      o = rs.getObject(i);
   }
}
```

测试示例中的 employee 表有 100000 条记录，每条记录平均长度 135 字节

以下是测试结果，对每种 fetchsize 测试 5 次再取平均值：

| fetchsize | elapse_time（s） |
|---|---|
| 1 | 20.516 |
| 2 | 11.34 |
| 4 | 6.894 |
| 8 | 4.65 |
| 16 | 3.584 |
| 32 | 2.865 |
| 64 | 2.656 |
| 128 | 2.44 |
| 256 | 2.765 |
| 512 | 3.075 |

| | |
|---|---|
| 1024 | 2.862 |
| 2048 | 2.722 |
| 4096 | 2.681 |
| 8192 | 2.715 |




Oracle jdbc fetchsize 默认值为 10，由上测试可以看出 fetchsize 对性能影响还是比较大的，但是当 fetchsize 大于 100 时就基本上没有影响了。fetchsize 并不会存在一个最优的固定值，因为整体性能与记录集大小及硬件平台有关。根据测试结果建议当一次性要取大量数据时这个值设置为 100 左右，不要小于 40。注意，fetchsize 不能设置太大，如果一次取出的数据大于 JVM 的内存会导致内存溢出，所以建议不要超过 1000，太大了也没什么性能提高，反而可能会增加内存溢出的危险。

注：图中 fetchsize 在 128 以后会有一些小的波动，这并不是测试误差，而是由于 resultset 填充到具体对像时间不同的原因，由于 resultset 已经到本地内存里了，所以估计是由于 CPU 的 L1,L2 Cache 命中率变化造成，由于变化不大，所以笔者也未深入分析原因。

iBatis 的 SqlMapping 配置文件可以对每个 SQL 语句指定 fetchsize 大小，如下所示：

```
<select id="getAllProduct" resultMap="HashMap" fetchSize="1000">
    select * from employee
</select>
```

四．**优化查询类型**

1.count 工作方式：

Count 分为两种工作方式：count(col/expr)和 count(*)

Count(col/expr)：代表统计表列名或者表达式有值（non-null）的次数；

Count(*):代表统计结果中的行数；

## 五．优化业务逻辑

要通过优化业务逻辑来提高性能是比较困难的，这需要程序员对所访问的数据及业务流程非常清楚。

举一个案例：

某移动公司推出优惠套参，活动对像为 VIP 会员并且 2010 年 1，2，3 月平均话费 20 元以上的客户。

那我们的检测逻辑为：

```
select avg(money) as avg_money from bill where phone_no='13988888888' and date between '201001' and '201003';
select vip_flag from member where phone_no='13988888888';
if avg_money>20 and vip_flag=true then
begin
  执行套参();
end;
```

如果我们修改业务逻辑为:

```
select avg(money) as  avg_money from bill where phone_no='13988888888' and date between '201001' and '201003';
if avg_money>20 then
begin
  select vip_flag from member where phone_no='13988888888';
  if vip_flag=true then
  begin
    执行套参();
  end;
end;
```

通过这样可以减少一些判断 vip_flag 的开销，平均话费 20 元以下的用户就不需要再检测是否 VIP 了。

如果程序员分析业务，VIP 会员比例为 1%，平均话费 20 元以上的用户比例为 90%，那我们改成如下：

```
select vip_flag from member where phone_no='13988888888';
if vip_flag=true then
begin
  select avg(money) as avg_money from bill where phone_no='13988888888' and date between '201001' and '201003';
  if avg_money>20 then
  begin
    执行套参();
  end;
end;
```

这样就只有 1%的 VIP 会员才会做检测平均话费，最终大大减少了 SQL 的交互次数。

以上只是一个简单的示例，实际的业务总是比这复杂得多，所以一般只是高级程序员更容易做出优化的逻辑，但是我们需要有这样一种成本优化的意识。

八.数据库安全

8.1MySQL 和 SQL 字符串限制长度漏洞分析

max_packet_size

这个东西是用来限制 mysql 客户端和服务器通信数据包的长度的，比如一个查询为“select * from user where 1”,那么这个长度仅仅几十个字节，所以不会超标。在绝大多情况下，我们很难会超过 mysql 的默认限制 1M（可以想象一下，1M 的 SQL 语句还是很长的）。这里插一句，看到这篇文章之后，我终于清楚我当初用 PEAR DB 的 INSERT 插入数据失败的原因了，很可能就是数据长度超标。对于 MySQL 来说，如果查询字符串的大小超过了这个限制，mysql 将不会执行任何查询操作

如果访问者有可能控制你的 sql 长度，那么你的程序可能会受到攻击。哪些情况访问者可能控制 sql 的长度呢，比如不限制关键字长度的搜索。还有可能就是你的程序如果要将用户的登录作为日志启用，总之凡是涉及到超长 sql 查询的地方，一定得小心检查自己的 sql，防止超长而查询失效。不过说实在的，本人认为这个问题倒不是多大，数据库光里管理员也可以自行设置 MySQL 的 max_packet_size 的长度，或者在处理可能超长的 SQL 查询的时候做一个长度判断。

MySQL 列长度限制

这个是本文的重点。MySQL 对于插入的字符串，如果长度超过了数据表限制的长度，MySQL 将会截取前面部分字符串插入数据库中，而不会将错误报给 web 程序。对于粗心的程序员，这个问题可能会导致程序的漏洞，其实目前的 wordpress 有很多限制，通过这个漏洞攻击应该没有任何作用。下面是原作者的几个假设，如果同时满足这几个条件，获取一个站点的用户名是相当容易的事情，幸运的是目前的 wordpress 并不太可能会同时满足下面的条件：

- 该 web 应用允许用户注册（开放注册的 wordpress 满足此条件）；
- 超级管理员的用户名已知的，比如 admin，这样方便攻击者寻找目标（可怜 wordpress 也满足）
- MySQL 使用的是默认的配置（估计大多数都满足）
- 注册新用户的时候，程序没有对用户名的长度给予限制（我测试过，wordpress 也满足）
- 用户名被限制在 16 个字符（这个和上面的没有关系，仅仅是方便举例）

下面我们来看看攻击者是怎么攻击的：

首先攻击者用已知的超级管理员 id 如 admin 注册，那么这个时候程序就会用

(show/hide)plain text

```
1.  SELECT * FROM user WHERE username='admin '
2.
```

来检查该 ID 是否已经存在，如果存在，这不允许注册，当然，攻击者尝试注册 admin 肯定会失败；

但是如果攻击者用 admin X（admin 和 x 之间有 11 个或以上的空格）来注册呢，按照上面的判断，由于 admin x 不存在数据库中，所以当然就能注册成功了，事实上 wordpress2.6.1 之前的版本确实可以这样，由于列长度的限制在 16 个字符内，所以末尾的 x 就被截掉了，那么现在数据库中就存在两个一模一样的用户 admin 了。(旁白：糟糕，那我的程序不是都要去修改。其实没有必要，你只要把 ID 设置为 UNIQUE 就可以了，于是乎，下面的问题就和你没有关系了)

攻击者继续，这个时候攻击者就顺利的注册了 admin 这个用户名，然后攻击者用 admin 和自己的密码登录进入账户管理（wordpress 即使注册了也无法登陆），由于真正的 admin 的帐号先于攻击者 admin 注册，所以在账户信息页面，显示的信息非常有可能就是真正 admin 的信息，包括密码提示和 email 等，这个时候攻击者就可以对 admin 的信息进行任意修改，包括密码和密码找回。

所以，写 web 程序的你，是不是该去检查一下自己的程序是否有此类的漏洞呢。

参考：


http://dba.taobao.net:9999/wiki/index.php/Main_Page

http://blog.csdn.net/yzsind/archive/2010/12/06/6059209.aspx

http://www.storyday.com/html/y2008/1898_mysql-and-sql-string-restrictions-on-the-length-of-the-loopholes.html