猜数字？

1,698,635,000

~~1,339,724,852~~

1,272,460,000

133.5%

507,634,000

263706000

192.5%

68

！本次活动奖品(棒棒糖)由素菜团特别赞助！

MySQL数据库架构与MySQL数据库开发的38条军规

吴诗展

www.ganji.com

~~MySQL数据库架构与~~ MySQL数据库开发的38条军规

吴诗展

www.ganji.com

MySQL数据库开发的38条军规

吴诗展

www.ganji.com

MySQL数据库架构与MySQL数据库开发的38条军规

吴诗展

www.ganji.com

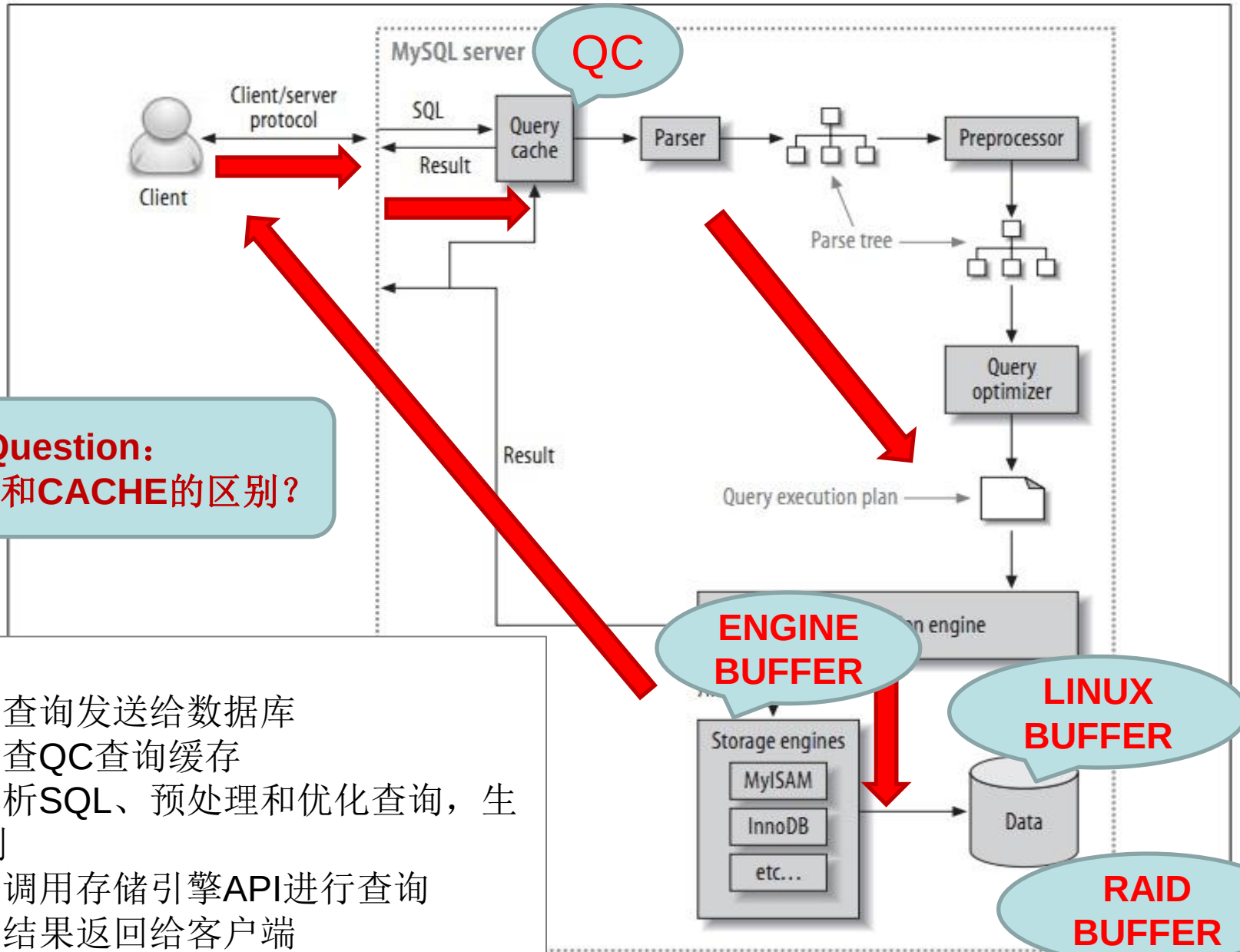
课程介绍

- 培训目的：
  - 主要讲授MySQL数据库开发中的一些技巧和注意点，并对MySQL架构体系进行介绍
- 培训对象：
  - 面向的学员是新入职的DBA
  - 使用MySQL的初中级RD
- 培训时间：
  - 全部课程大约80分钟

0 MySQL架构概述

www.ganji.com

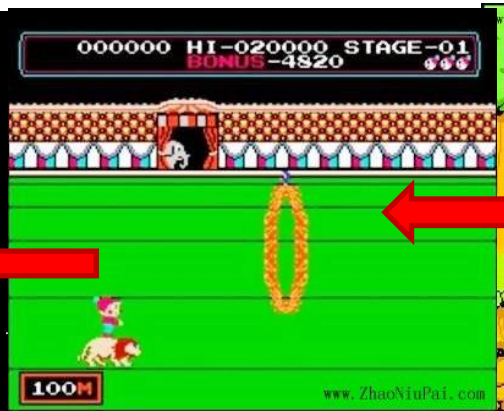
一封邮件：A select ...

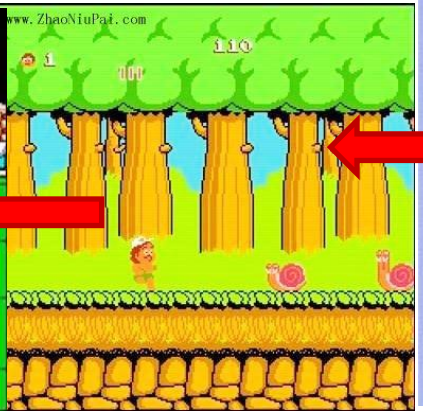


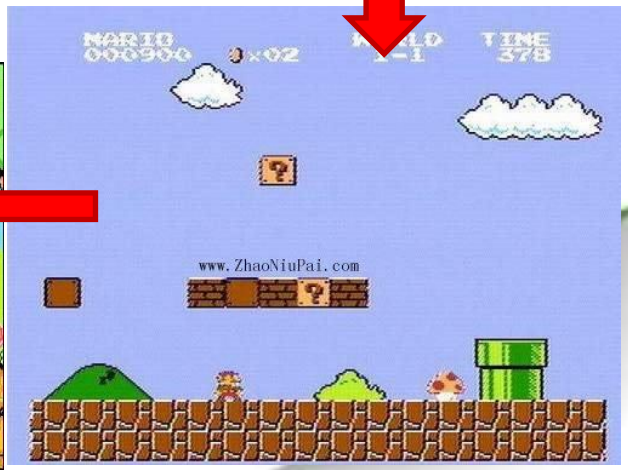
Question：
BUFFER和CACHE的区别？

五大步骤：
1.客户端将查询发送给数据库
2.数据库检查QC查询缓存
3.数据库解析SQL、预处理和优化查询，生成执行计划
4.执行引擎调用存储引擎API进行查询
5.服务器将结果返回给客户端

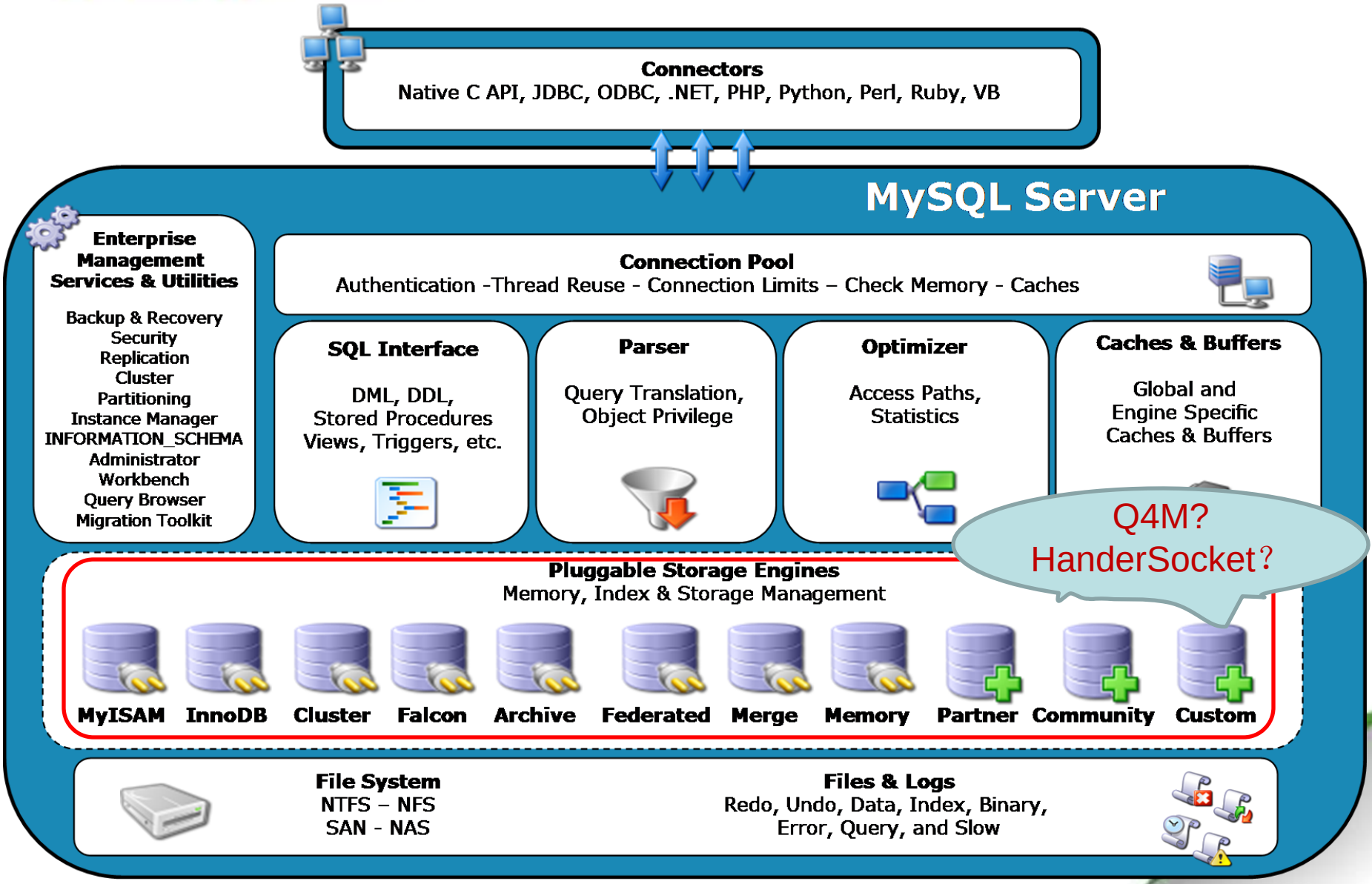
Pluggable Engine Architecture







可插式存储引擎架构体系

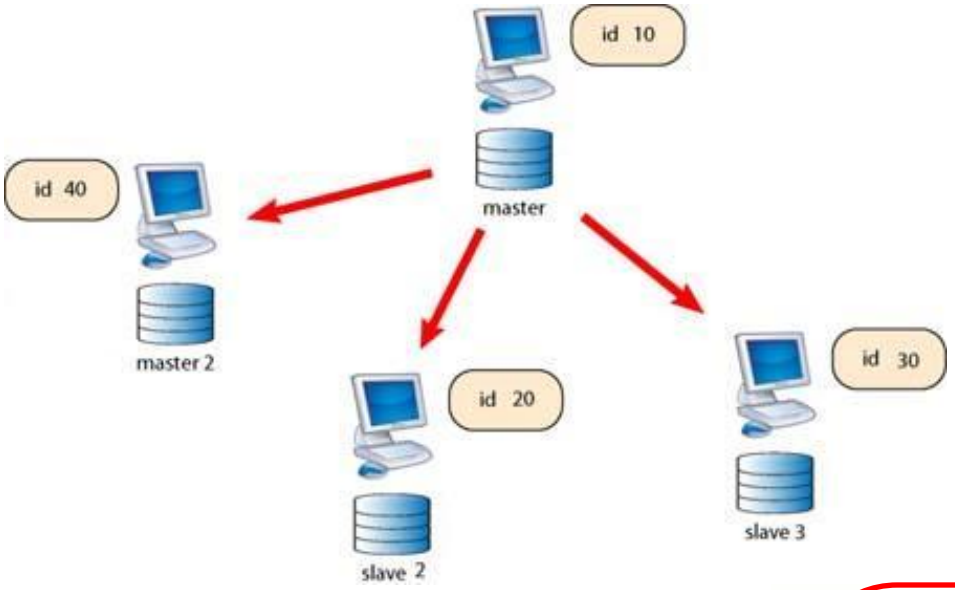


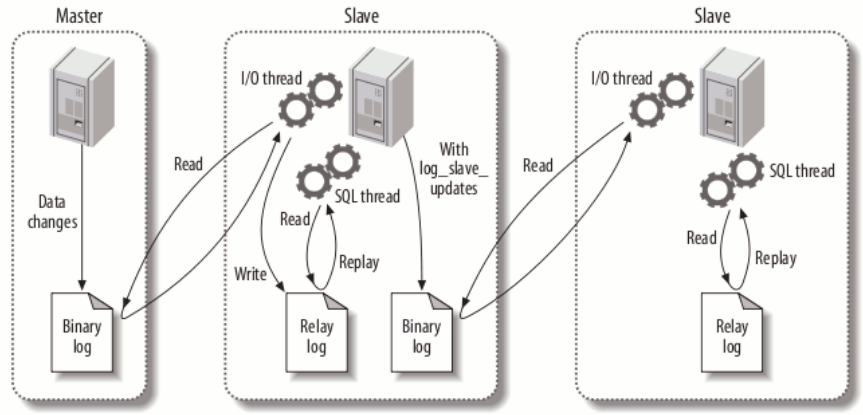
Replication 1→N






MySQL Replication



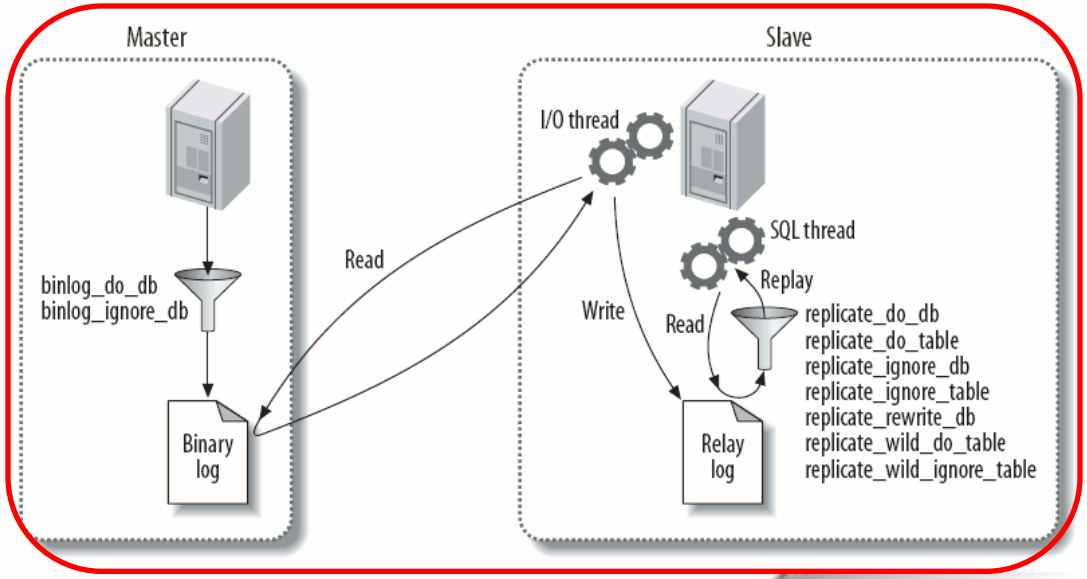


SQL复制过程：

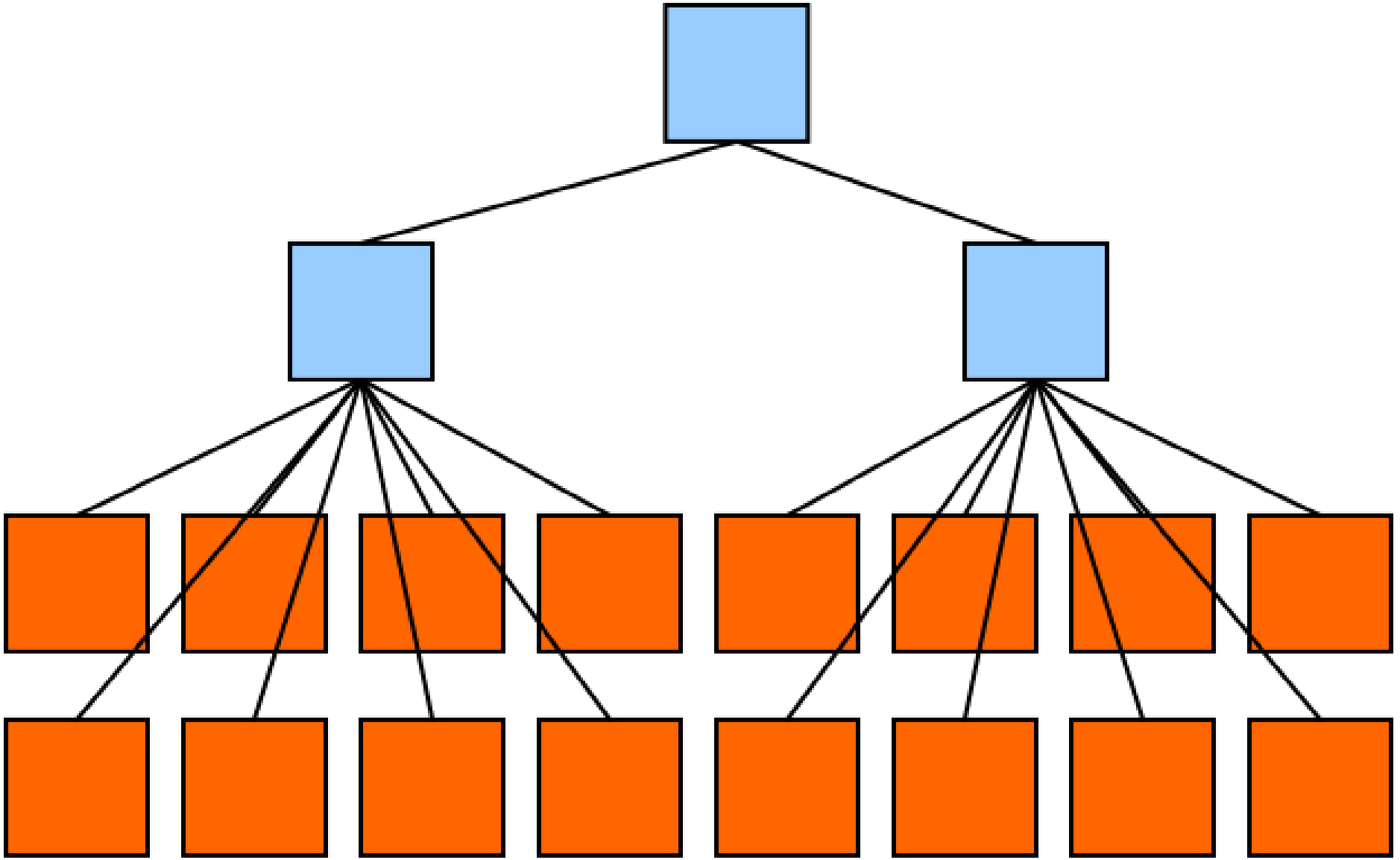
1.主库执行SQL
2.SQL写到Binlog
3.从库IO Thread 抓取Binlog
4.从库SQL Thread 执行

异步复制：

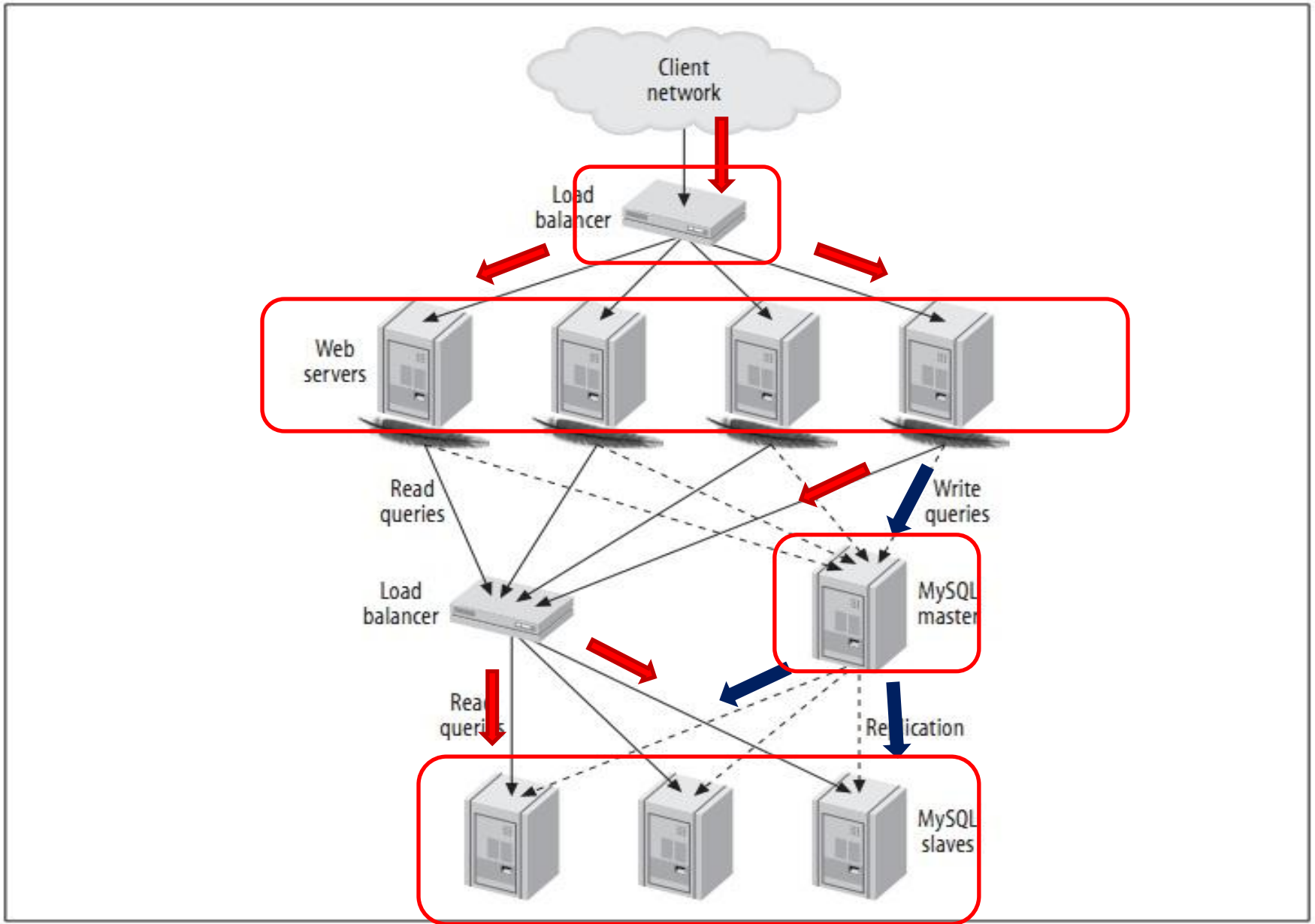
~~大SQL，大事务，大批量~~



MySQL Replication



典型WEB2.0网站架构



架构体系小结

- 一条查询实现过程
- 小霸王与可插式存储引擎架构
- 孙猴子与MySQL Replication
- 典型WEB2.0网站架构

----“军规”来也。。。

1 基础军规

尽量不在数据库做运算

- 别让脚趾头想事情
- 那是脑瓜子的职责

- 让数据库多做她擅长的事：
  - ✓ 尽量不在数据库做运算
  - ✓ 复杂运算移到程序端CPU
  - ✓ 尽可能简单应用MySQL

- 举例: ~~md5() / Order by Rand()~~

控制单表数据量

- 一年内的单表数据量预估
  - 纯INT不超1000W
  - 含CHAR不超500W

- 合理分表
  - USERID
  - DATE
  - AREA
  - ….
- 建议单库不超过300-400个表

保持表身段苗条

- 表字段小而精
  - √ IO高效 √全表遍历 √表级锁
  - √提高并发 √alter table快
  - √损坏修复快
- 顺序读1G数据需 3~10 秒！
- 单表500W行1G体积评估
  - ➢ 单行不超过200Byte
  - ➢ 单表不超50个纯INT字段
  - ➢ 单表不超20个CHAR(10)字段
- **字段数上限控制在20~50个**

```
Create Table: CREATE TABLE `user_company` (
  `id` int(10) NOT NULL AUTO_INCREMENT COMMENT '公司的id',
  `user_id` int(10) NOT NULL COMMENT '用户id',
  `name` varchar(100) NOT NULL COMMENT '公司名字',
  `latlng` varchar(100) NOT NULL COMMENT '经纬度',
  `contact` varchar(50) NOT NULL COMMENT '联系人',
  `contact_phone` varchar(50) NOT NULL COMMENT '联系电话',
  `contact_mobile` varchar(50) NOT NULL COMMENT '手机',
  `email` varchar(50) NOT NULL COMMENT '邮箱地址',
  `biz_type` varchar(100) NOT NULL COMMENT '公司的行业',
  `company_type` int(11) NOT NULL,
  `tuiguang_type` int(11) NOT NULL,
  `scale` int(11) NOT NULL,
  `address` varchar(100) NOT NULL COMMENT '公司所在地址',
  `description` text NOT NULL COMMENT '公司描述信息',
  `license_num` varchar(100) NOT NULL COMMENT '营业执照&资格证号码',
  `license` varchar(100) NOT NULL COMMENT '营业执照照片的url',
  `license_thumb` varchar(100) NOT NULL COMMENT '营业执照照片的缩略图',
  `company_header` varchar(100) DEFAULT NULL COMMENT '公司图片',
  `province` int(11) NOT NULL,
  `city` int(11) NOT NULL,
  `is_licensed` int(11) NOT NULL,
  `licence_audit_remark` varchar(512) NOT NULL COMMENT '审核营业执照后的原因',
  `refused_id` int(11) NOT NULL,
  `refused_reason` varchar(100) NOT NULL COMMENT '拒绝原因描述',
  `auditor` varchar(100) NOT NULL COMMENT '审核人',
  `audit_time` int(10) NOT NULL COMMENT '审核时间',
  `auth_type` int(11) NOT NULL,
  `sale_id` int(10) unsigned DEFAULT NULL,
  `sale_name` varchar(20) DEFAULT NULL,
  `is_notified` int(11) NOT NULL,
  `website` varchar(100) DEFAULT NULL COMMENT '公司网址',
  `industry_type` int(11) NOT NULL,
  `factname_status` int(11) NOT NULL,
  `factname_count` int(11) NOT NULL,
  `bank_auth_status` int(11) NOT NULL,
  `idcard` varchar(50) NOT NULL COMMENT '身份证',
  `idcard_auth_type` int(11) NOT NULL,
  `idcard_image` varchar(100) NOT NULL COMMENT '身份证图片',
  `idcard_confirm_status` int(11) NOT NULL,
  `bank_type` int(10) NOT NULL COMMENT '银行id',
  `bank_province_id` int(10) NOT NULL COMMENT '银行省id',
  `bank_province_name` varchar(50) NOT NULL COMMENT '银行省名称',
  `bank_city_id` int(10) NOT NULL COMMENT '银行城市id',
  `bank_city_name` varchar(50) NOT NULL COMMENT '银行城市名称',
  `bank_type_name` varchar(50) NOT NULL COMMENT '银行名称',
  `bank_name` varchar(255) NOT NULL,
  `bank_account` varchar(50) NOT NULL COMMENT '银行账号',
  `bank_input_method` int(11) NOT NULL,
  `bank_confirm_status` int(11) NOT NULL,
  `bank_confirm_count` int(11) NOT NULL,
  `bank_confirm_amount1` decimal(3,2) NOT NULL COMMENT '银行用户确认金额1',
  `bank_confirm_amount2` decimal(3,2) NOT NULL COMMENT '银行用户确认金额2',
  `is_underline` int(11) NOT NULL,
  `create_time` int(10) NOT NULL COMMENT '创建时间',
  `update_time` int(10) NOT NULL COMMENT '更新时间',
  `xsjz_submit_time` int(10) NOT NULL COMMENT '学生兼职认证提交时间，若第二次提交，不更新此时间',
  PRIMARY KEY (`id`),
  KEY `user_id` (`user_id`),
  KEY `name` (`name`),
  KEY `province` (`province`),
  KEY `city` (`city`),
  KEY `factname_status` (`factname_status`),
  KEY `email` (`email`),
  KEY `industry_type` (`industry_type`),
  KEY `idx_tuiguang_type` (`tuiguang_type`),
  KEY `sale_id` (`sale_id`),
  KEY `idx_bank_auth_status_is_licensed` (`bank_auth_status`,`is_licensed`),
  KEY `idx_bank_account` (`bank_account`)
) ENGINE=InnoDB AUTO_INCREMENT=4091468 DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

拒绝3B

- 拒绝3B
  - 大SQL (**B**IG SQL)
  - 大事务 (**B**IG Transaction)
  - 大批量 (**B**IG Batch)

- 详细解析下文讲述

基础军规小结

- 尽量不在数据库做运算
- 控制单表数据量
- 保持表身段苗条
- 拒绝3B

2 字段类军规

用好数值字段类型

- 三类数值类型：
  - ~~✓ TINYINT(1Byte)、SMALLINT(2B)、MEDIUMINT(3B)~~
  - ✓ INT(4B)、BIGINT(8B)
  - ✓ FLOAT(4B)、DOUBLE(8B)
  - ✓ DECIMAL(M,D)

BAD CASE：
- INT(1) VS INT(11)
- BIGINT AUTO_INCREMENT
- NO UNSIGNED used
- DECIMAL(31,0)

- 赶集内部军规：整型统一用INT

将字符转化为数字

- 数字型VS字符串型索引
  - ✓ 更高效
  - ✓ 查询更快
  - ✓ 占用空间更小

- 举例：存储IP用INT，而非CHAR
  - ➢ INT UNSIGNED
  - ➢ INET_ATON()
  - ➢ INET_NTOA()

IP 地址存储举例

```
CREATE TABLE Sessions (
  session_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, ip_address INT UNSIGNED NOT NULL // 对比 CHAR(15)!!
, session_data TEXT NOT NULL
, PRIMARY KEY (session_id)
, INDEX (ip_address)
) ENGINE=InnoDB;
```

```
// 找本地IP 的session
SELECT * FROM Sessions
WHERE ip_address BETWEEN
INET_ATON('192.168.0.1') AND
INET_ATON('192.168.0.255');
```

INET_ATON()转化为数值，极大提高查询效率:

```
SELECT * FROM Sessions
WHERE ip_address BETWEEN 3232235521 AND 3232235775
```

用INT存储日期

```
CREATE TABLE Sessions (
  session_id INT UNSIGNED NOT NULL AUTO_INCREMENT
, create_time  INT UNSIGNED NOT NULL
, session_data TEXT NOT NULL
, PRIMARY KEY (session_id)
, INDEX (ip_address)
) ENGINE=InnoDB;
```

```
// 统计1月份的session
SELECT  * FROM Sessions
WHERE create_time BETWEEN
unix_timestamp('2011-01-01') AND unix_timestamp('2011-01-31 23:59:59‘);  //反解用from_unixtime()
```

unix_timestamp转化为数值，极大提高查询效率:

```
SELECT * FROM Sessions
WHERE create_time BETWEEN 1293811200 AND  1296489599
```

优先使用ENUM或SET

- 优先使用ENUM或SET
  - 字符串
  - 可能值已知且有限
- 存储
  - ENUM存储1字节，转为数值运算
  - SET视节点定，最多存储8字节
  - 比较时需要加 ‘ 单引号(即使是数值)
- 举例
  - `sex` enum('F','M') COMMENT '性别'
  - `c1` enum('0','1','2','3') COMMENT '职介审核'

避免使用NULL字段

- 避免使用NULL字段
  - 很难进行查询优化
  - NULL列加索引，需要额外空间
  - 含NULL复合索引无效

- 举例
  - ~~`a` char(32) DEFAULT NULL~~
  - ~~`b` int(10) NOT NULL~~
  - `c` int(10) NOT NULL DEFAULT 0

少用并拆分TEXT/BLOB

- TEXT类型处理性能远低于VARCHAR
  - 强制生成硬盘临时表
  - 浪费更多空间
  - VARCHAR(65535)==>64K
- 尽量不用TEXT/BLOB数据类型
- 若必须使用则拆分到单独的表

不在数据库里存图片

post_image_id	created_time	file_bytes	
1	2010-08-04 16:46:53	(Binary/Image)	39K
2	2010-08-04 16:46:55	(Binary/Image)	37K
3	2010-08-04 16:46:58	(Binary/Image)	70K
4	2010-08-04 19:18:34	(Binary/Image)	76K
5	2010-08-05 12:06:39	(Binary/Image)	34K
6	2010-08-05 12:06:40	(Binary/Image)	41K
7	2010-08-05 12:09:42	(Binary/Image)	48K
8	2010-08-05 14:21:14	(Binary/Image)	167K
9	2010-08-05 14:21:15	(Binary/Image)	197K
10	2010-08-05 14:21:16	(Binary/Image)	133K
11	2010-08-05 14:28:23	(Binary/Image)	64K
12	2010-08-05 14:33:38	(Binary/Image)	34K
13	2010-08-05 14:33:39	(Binary/Image)	41K
14	2010-08-05 14:36:17	(Binary/Image)	167K


```
(admin@ms)[(none)]> select count(*) from post_data_cache.post_image_data;
+----------+
| count(*) |
+----------+
|   383676 |
+----------+
1 row in set (0.00 sec)

(admin@ms)[(none)]> quit
Bye
[work@yz-off-ku-real00 post_data_cache]$ ll -h post_image_data*
-rw-rw---- 1 work work 8.8K Dec 29 00:27 post_image_data.frm
-rw-rw---- 1 work work  23G Jun 23 17:11 post_image_data.MYD
-rw-rw---- 1 work work 7.9M Jun 23 17:11 post_image_data.MYI
```

字段类军规小结

- 用好数值字段类型
- 将字符转化为数字
- 用INT存储日期
- 多用枚举ENUM/SET
- 避免将字段设为NULL
- 少用并拆分TEXT/BLOB
- 不在数据库里存图片

3 索引类军规

谨慎合理添加索引

- 谨慎合理添加索引
  - 改善查询
  - 减慢更新
  - 索引不是越多越好

- 能不加索引的尽量不加
  - 最好不超过字段数20%

- 举例
  - 不要给“性别”列创建索引

不在索引列做运算

- 不在索引列进行数学运算或函数运算
  - 无法使用索引
  - 导致全表扫描

- 举例

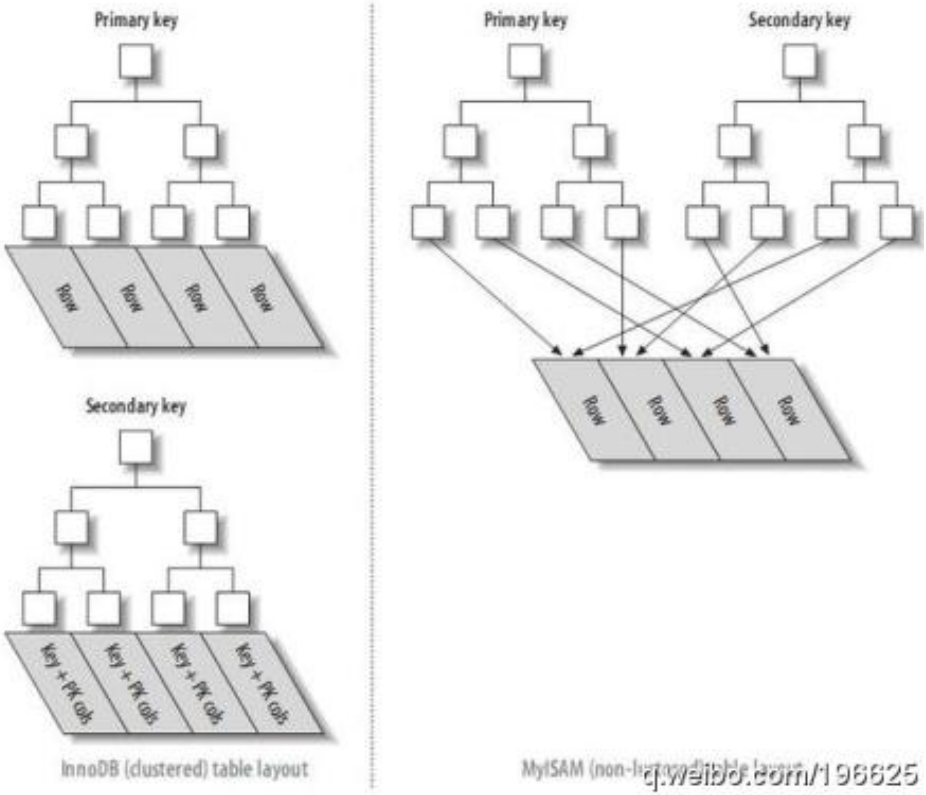
```
(admin@ms)[beijing]> select * from wanted_post where id +1 = 6630913 \G
1 row in set (51.62 sec)

(admin@ms)[beijing]> select * from wanted_post where id    = 6630913 -1 \G
1 row in set (0.00 sec)
```

SELECT count(*) FROM findjob_post WHERE findjob_open_status <= 1 AND editor_audit_status = 1 AND findjob_lastmodify > unix_timestamp('2010-11-29')-86400;

选择自增INT做INNODB主键

- 对主键建立聚簇索引
- 二级索引存储主键值
- 主键不更新修改
- 按主键顺序插入值
- 忌用字符串做主键
- 推荐用AUTO_INCREMENT列做主键
- 若不指定主键，InnoDB会用唯一且非空值索引代替



尽量不用外键

- 线上OLTP系统（线下统计另论）
  - 可节省开发量
  - 有额外开销
  - 可‘到达’其它表，意味着锁
  - 高并发时容易死锁
- 由程序保证约束

字符字段必须建前缀索引

- 字符字段建前缀索引
  - 26个字母
  - 26*26*26*26=456,976
  - 26*26*26*26*26=11,881,376
  - 26*26*26*26*26*26=308,915,776

- 字符字段必须建前缀索引

```
`pinyin` varchar(100) DEFAULT NULL COMMENT '小区拼音',
 KEY `idx_pinyin` (`pinyin`(8)),
) ENGINE=InnoDB  COMMENT='房产推广汇总表'
```

索引类军规小结

- 谨慎合理添加索引
- 不在索引列做运算
- 选择自增INT做INNODB主键
- 尽量不用外键
- 字符字段必须建前缀索引

ganji 赶集

4 SQL类军规

SQL语句越简单越好

- 大SQL VS 多个简单SQL
  - 传统设计思想
  - BUT MySQL NOT
  - 一条SQL只能在一个CPU运算
  - 2000+ QPS的高并发中，1秒大SQL意味着？
  - **可能一条大SQL就把整个数据库堵死**
- 拒绝大SQL，拆解成多条简单SQL
  - 简单SQL缓存命中率更高
  - 减少锁表时间，特别是MyISAM
  - 用上多CPU

保持事务(连接)短小

- 保持事务(连接)短小
  - 即开即用，用完即关
  - 与事务无关操作放到事务外面, 减少锁资源的占用
  - 不破坏一致性前提下，使用多个短事务代替长事务
- 举例
  - 图片的上传
  - 大量的sleep连接

尽可能避免使用SP/TRIG/FUNC

- 线上OLTP系统（线下统计再论）
  - 尽可能少用存储过程
  - 尽可能少用触发器
  - 减用使用MySQL函数对结果进行处理
- 交给客户端程序负责

不用 SELECT *

- 用SELECT *
  - 更多消耗CPU、内存、IO、网络带宽
  - 先向数据库请求不需要的列，然后丢掉？
- 不用SELECT * ，只取需要数据列
  - 更安全的设计：减少表变化带来的影响
  - 使用covering index提供可能性
  - Select/JOIN减少硬盘临时表生成，特别是有TEXT/BLOB时
- 举例

SELECT * FROM tag WHERE id = 999184

➔

SELECT keyword FROM tag WHERE id = 999184

UPDATE的WHERE子句要有索引

- UPDATE的WHERE子句要有索引，否则会
  - 导致全表扫描
  - 导致主从延迟

•举例

major_category_statisti引起的从库延迟故障通报

故障时间段	201012.08 02:00- 10:16	**操作人/负责人**	
故障影响	主站数据库延迟。		
处理方式	脚本运行结束后恢复正常。		
改进措施	添加相关索引，执行时间由0.2秒变为0.002秒左右，第二天未再出现同类问题。		
其它说明	说明： SQL：UPDATE ganji.major_category_statistic SET statisdate=1296057600,statiscount=1 WHERE city_id=268 AND category_url='qzzagong' 是用于首页类别统计帖子数的，昨天将相关脚本由test调整到线上，但由于update SQL没有索引，每次update都走全表，需要 0.2秒到0.26秒左右，总约152227条，这个脚本是昨天2点开始运行的，今天早上10点16分左右结束，完成这些更新约需8-10小时，即机器的时间耗费到此SQL上了，导致从库**延迟。**		

```
(root@ms)[(none)]> UPDATE ganji.major_category_statistic SET statisdate=1296057600,statiscount=1  WHERE city_id=268 AND category_url
='qzzagong'  ;
Query OK, 0 rows affected (0.25 sec)
Rows matched: 4  Changed: 0  Warnings: 0

(root@ms)[(none)]> UPDATE ganji.major_category_statistic SET statisdate=1296057600,statiscount=1  WHERE city_id=268 AND category_url
='qzzagong'  ;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 4  Changed: 0  Warnings: 0
```

改写OR为IN()

- 同一字段，将or改写为in()
  - OR效率：O(n)
  - IN 效率：O(Log n)
  - 当n很大时，OR会慢很多
- 注意控制IN的个数，建议n小于200
- 举例

Select * from opportunity WHERE phone= ‘123456' or phone= ‘422422' \G

➔

Select * from opportunity WHERE phone in ('123456' , '422422')

改写OR为UNION

- 不同字段，将or改为union
  - 减少对不同字段进行 "or" 查询
  - Merge index往往很弱智
  - 拆分成多个单一字段查询SQL效率更高

- 举例

```
Select * from  opportunity WHERE phone='021-64347221' or cellPhone='13917485584' \G
➔
Select * from  opportunity WHERE phone='021-64497221'
union
Select * from  opportunity WHERE   cellPhone='13967485584' \G
```

避免 % 前缀模糊查询

- 避免 % 前缀模糊查询
  - B+ Tree
  - 使用不了索引
  - 导致全表扫描

- 举例

MySQL> select * from wanted_post WHERE title like '亦庄%' ;
298 rows in set (**0.01** sec)

MySQL> select * from wanted_post WHERE title like '**%**亦庄%' ;
572 rows in set (**3.27** sec)

LIMIT的高效分页

- 传统分页：
  - Select * from table limit 10000,10;
- LIMIT原理：
  - Limit 10000,10
  - 偏移量越大则越慢
- 推荐分页：
  - Select * from table WHERE id>=23423 limit 11;
    #10+1
  - select * from table WHERE id>=23424 limit 11;

LIMIT的高效分页

- 分页方式二：
  - Select * from table WHERE id >= (select id from table limit 10000,1) limit 10;
- 分页方式三：
  - SELECT * FROM table INNER JOIN (SELECT id FROM table LIMIT 10000,10) USING (id) ;
- 分页方式四：
  - 程序取ID：select id from table limit 10000,10;
  - Select * from table WHERE id in (123,456…) ;
- 可能需按场景分析并重组索引

LIMIT的高效分页

- 示例：

MySQL> select sql_no_cache * from wanted_post limit 10,10;
10 row in set (**0.01** sec)

MySQL> select sql_no_cache * from wanted_post limit 20000,10;
10 row in set (**0.13** sec)

MySQL> select sql_no_cache * from wanted_post limit **80000,**10;
10 rows in set (**0.58** sec)

MySQL> select sql_no_cache id from wanted_post limit **80000,**10;
10 rows in set (**0.02** sec)

MySQL> select sql_no_cache * from wanted_post WHERE id>=323423 limit 10;
10 rows in set (**0.01** sec)

MySQL> select * from wanted_post WHERE id >= (select sql_no_cache id from wanted_post limit 80000,1) limit 10 ;
10 rows in set (**0.02** sec)

减少COUNT(*)

- 几个有趣的例子：
  - COUNT(COL) VS COUNT(*)
  - COUNT(*) VS COUNT(1)
  - COUNT(1) VS COUNT(0) VS COUNT(100)

```
`id` int(10) NOT NULL
AUTO_INCREMENT
COMMENT '公司的id',

`sale_id` int(10)
unsigned DEFAULT NULL,
```

- 示例

```
mysql> select count(*),count(0),
    -> count(1),count(100),count(id
    -> count(sale_id),count(null)
    -> from user_company\G
*************************** 1. row
      count(*): 4091458
      count(0): 4091458
      count(1): 4091458
    count(100): 4091458
     count(id): 4091458
count(sale_id): 403643
   count(null): 0
```

- 结论
  - ✓ COUNT(*)=count(1)
  - ✓ COUNT(0)=count(1)
  - ✓ COUNT(1)=count(100)
  - ✓ COUNT(*)!=count(col)
  - ✓ WHY?

减少COUNT(*)

- MyISAM VS INNODB
  - ✓ 不带 WHERE COUNT()
  - ✓ 带 WHERE　COUNT()
- COUNT(*)的资源开销大，尽量~~不用~~少用
- 计数统计
  - ✓ 实时统计：用memcache，双向更新，凌晨跑基准。
  - ✓ 非实时统计：尽量用单独统计表，定期计算

用UNION ALL 而非 UNION

- 若无需对结果进行去重，则用UNION ALL
  - UNION有去重开销

- 举例

```
MySQL>SELECT * FROM detail20091128 UNION ALL
SELECT * FROM detail20110427 UNION ALL
SELECT * FROM detail20110426 UNION ALL
SELECT * FROM detail20110425 UNION ALL
SELECT * FROM detail20110424 UNION ALL
SELECT * FROM detail20110423;
```

分解联接保证高并发

- 不建议进行两个表以上的JOIN
- 适当时分解联接保证高并发
  - ✓ 可缓存大量早期数据
  - ✓ 使用了多个MyISAM表
  - ✓ 对大表的小ID IN()
  - ✓ 联接引用同一个表多次
- 举例：

```
MySQL> 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= ‘二手玩具’;
➔
MySQL> Select *  from tag WHERE tag= ‘二手玩具’;
MySQL> Select * from tag WHERE tag_id=1321;
```

GROUP BY 去除排序

- GROUP BY 实现
  - ✓ 分组
  - ✓ 自动排序
- 无需排序：Order by NULL
- 特定排序：Group by DESC/ASC
- 举例

MySQL> select phone,count(*) from wanted_post group by phone limit 1 ;
1 row in set (2.19 sec)

MySQL> select phone,count(*) from wanted_post group by phone order by null limit 1;
1 row in set (2.02 sec)

同数据类型的列值比较

- 原则：数字比数字，字符比字符

- 数值列与字符类型比较
  - 同时转换为双精度
  - 进行比对

- 字符列与数值类型比较
  - 字符列整列转数值
  - 不会使用索引查询

同数据类型的列值比较

- 举例：字符列与数值类型比较

字段：`remark` varchar(200) NOT NULL COMMENT '备注,默认为空',

MySQL>SELECT `id`, `gift_code` FROM groupon_coupon_pool_gift
WHERE `deal_id` = 640 AND remark=115127;
1 row in set **(0.14** sec)

MySQL>SELECT `id`, `gift_code` FROM groupon_coupon_pool_gift
WHERE `deal_id` = 640 AND remark='115127';
1 row in set **(0.005** sec)

思考：若JOIN时比较的类型不匹配?

批量数据快导入

- 批量数据快导入：
  - 成批装载比单行装载更快，不需要每次刷新缓存
  - 无索引时装载比索引装载更快
  - Insert values ,values，values 减少索引刷新
  - Load data比insert快约20倍

- 尽量不用 INSERT … SELECT
  - 延迟
  - 同步出错

打散大批量更新

- 大批量更新凌晨操作，避开高峰
- 凌晨不限制、白天上限默认为100条/秒
- 举例：

```
update post set tag=1 WHERE  id in (1,2,3);
sleep 0.01;
update post set tag=1 WHERE  id in (4,5,6);
sleep 0.01;
……
```

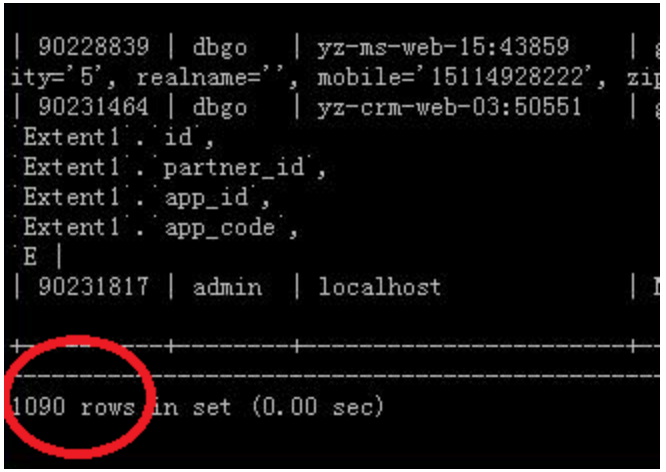
SQL类军规小结

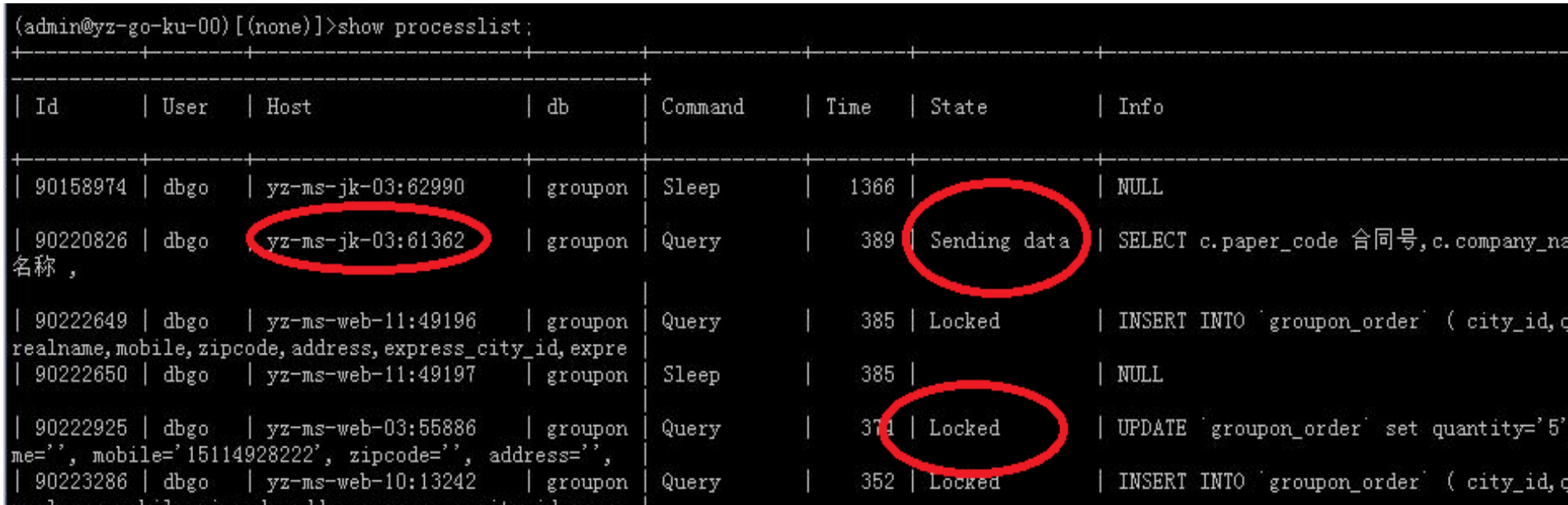
- SQL语句越简单越好
- 保持事务(连接)短小
- 尽可能避免使用SP/TRIG/FUNC
- 不用 SELECT *
- UPDATE的WHERE子句要用索引
- 改写OR为IN
- 改写OR为UNION
- 避免 % 前缀模糊查询
- LIMIT的高效分页
- 减少COUNT(*)
- 用UNION ALL 而非 UNION
- 分解联接保证高并发
- GROUP BY 去除排序
- 同数据类型的列值比较
- 批量数据快导入
- 打散大批量更新

5 约定类军规

隔离线上线下

- 线上连线上，线下连线下原则
  - 实时数据用real库
  - 模块环境用sta库
  - 测试用qa库
- 九号团购案例：





禁止未经DBA确认的子查询

- MySQL子查询
  - 大部分情况优化较差
  - 特别WHERE中使用IN id的子查询
  - 一般可用JOIN改写

- 举例:

~~MySQL> insert into table1 (select * from table2);~~

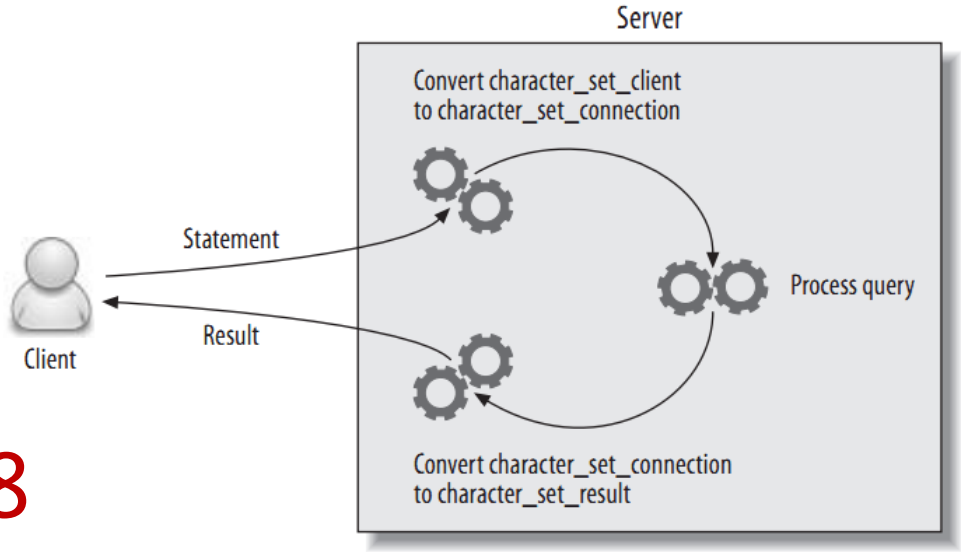
永远不在程序端显式加锁

- 永远不在程序端显式加锁
  - 外部锁对数据库不可控
  - 高并发时是灾难
  - 极难调试和排查

- 并发扣款等一致性问题
  - 采用事务
  - 相对值修改
  - Commit前二次较验冲突

统一字符集为UTF8

- 字符集：
  - MySQL 4.1 以前只有latin1
  - 为多语言支持增加多字符集
  - 也带来了N多问题
  - 保持简单



- 统一字符集为UTF8
- 乱码：SET NAMES UTF8

库表命名统一用小写

- MySQL库表大小写敏感性
  - Linux VS Windows
  - 库表名称统一用小写
  - 字段不区分大小写

- 库名用缩写，尽量在2~7个字母
  - DataSharing ==> ds

避免用保留字命名

- 举例：~~Select * from return;~~

Select * from `return`;

ADD	ALL	ALTER	GOTO	GRANT	GROUP	PURGE	RAID0	RANGE
ANALYZE	AND	AS	HAVING	HIGH_PRIORITY	HOUR_MICROSECOND	READ	READS	REAL
ASC	ASENSITIVE	BEFORE	HOUR_MINUTE	HOUR_SECOND	IF	REFERENCES	REGEXP	RELEASE
BETWEEN	BIGINT	BINARY	IGNORE	IN	INDEX	RENAME	REPEAT	REPLACE
BLOB	BOTH	BY	INFILE	INNER	INOUT	REQUIRE	RESTRICT	RETURN
CALL	CASCADE	CASE	INSENSITIVE	INSERT	INT	REVOKE	RIGHT	RLIKE
CHANGE	CHAR	CHARACTER	INT1	INT2	INT3	SCHEMA	SCHEMAS	SECOND_MICROSECOND
CHECK	COLLATE	COLUMN	INT4	INT8	INTEGER	SELECT	SENSITIVE	SEPARATOR
CONDITION	CONNECTION	CONSTRAINT	INTERVAL	INTO	IS	SET	SHOW	SMALLINT
CONTINUE	CONVERT	CREATE	ITERATE	JOIN	KEY	SPATIAL	SPECIFIC	SQL
CROSS	CURRENT_DATE	CURRENT_TIME	KEYS	KILL	LABEL	SQLEXCEPTION	SQLSTATE	SQLWARNING
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR	LEADING	LEAVE	LEFT	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT
DATABASE	DATABASES	DAY_HOUR	LIKE	LIMIT	LINEAR	SSL	STARTING	STRAIGHT_JOIN
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND	LINES	LOAD	LOCALTIME	TABLE	TERMINATED	THEN
DEC	DECIMAL	DECLARE	LOCALTIMESTAMP	LOCK	LONG	TINYBLOB	TINYINT	TINYTEXT
DEFAULT	DELAYED	DELETE	LONGBLOB	LONGTEXT	LOOP	TO	TRAILING	TRIGGER
DESC	DESCRIBE	DETERMINISTIC	LOW_PRIORITY	MATCH	MEDIUMBLOB	TRUE	UNDO	UNION
DISTINCT	DISTINCTROW	DIV	MEDIUMINT	MEDIUMTEXT	MIDDLEINT	UNIQUE	UNLOCK	UNSIGNED
DOUBLE	DROP	DUAL	MINUTE_MICROSECOND	MINUTE_SECOND	MOD	UPDATE	USAGE	USE
EACH	ELSE	ELSEIF	MODIFIES	NATURAL	NOT	USING	UTC_DATE	UTC_TIME
ENCLOSED	ESCAPED	EXISTS	NO_WRITE_TO_BINLOG	NULL	NUMERIC	UTC_TIMESTAMP	VALUES	VARBINARY
EXIT	EXPLAIN	FALSE	ON	OPTIMIZE	OPTION	VARCHAR	VARCHARACTER	VARYING
FETCH	FLOAT	FLOAT4	OPTIONALLY	OR	ORDER	WHEN	WHERE	WHILE
FLOAT8	FOR	FORCE	OUT	OUTER	OUTFILE	WITH	WRITE	X509
FOREIGN	FROM	FULLTEXT	PRECISION	PRIMARY	PROCEDURE	XOR	YEAR_MONTH	ZEROFILL

约定类军规小结

- 隔离线上线下
- 禁止未经DBA确认的子查询上线
- 永远不在程序端显式加锁
- 统一字符集为UTF8
- 库表命名统一用小写
- 避免用保留字命名

军规小结

- 一.基础军规(4)
- 二.字段类军规(7)
- 三.索引类军规(5)
- 四.SQL类军规(16)
- 五.约定类军规(6)

谢谢

尽量简单去用数据库，让她做擅长的工作！

希望数据库像钻石般尊贵、简洁、稳定！

问题解答

推荐资源

- 推荐资源：
  - MySQL Manual
  - MySQL Internals Manual
  - MySQLperformanceblog.com
  - 《High Performance MySQL》
  - 《MySQL性能调优与架构设计》