MySQL
开发规范与SQL优化

目录

MySQL 索引

什么是索引

●select * from Score where score=“77” ;

id,name,class,...,...,...,score,desc,date,
id,name,class,...,...,...,score,desc,date,
id,name,class,...,...,...,score,desc,date,
让你实现在1,000,000行文本文件中查找你会怎么做？

- for(String line : lines){
  String[] words = line.split(",");
  for(String word : words){
  if(word.equals(77)){
  System.out.println(line);
  }
  }
- } 一行一行扫描（全表扫描）？太慢，黄花菜都凉了。

索引结构

- Btree ?
- Hash?
- Bitmap?

为什么使用B-TREE（B+TREE）

- 一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘I/O消耗，相对于内存存取，I/O存取的消耗要高几个数量级，所以评价一个数据结构作为索引的优劣最重要的指标就是在查找过程中磁盘I/O操作次数的渐进复杂度。

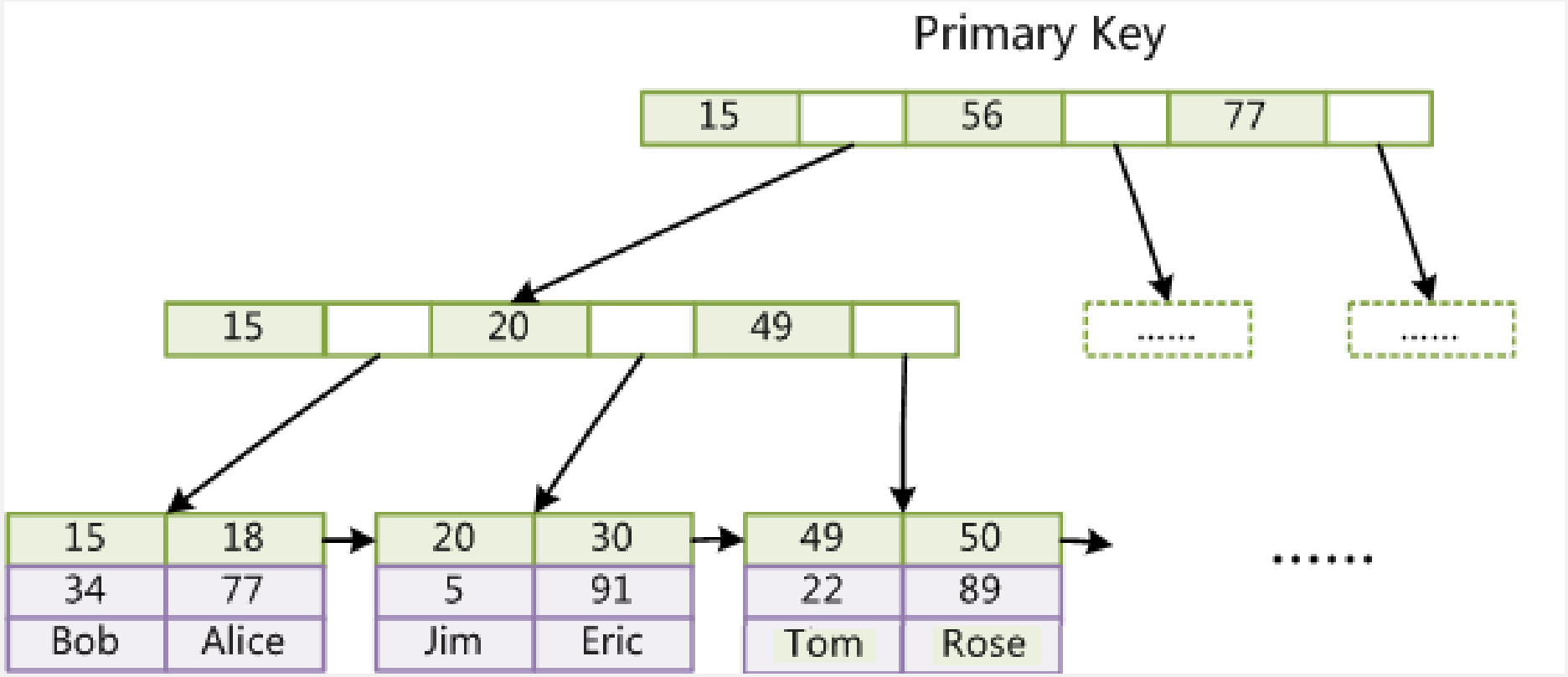
为什么使用B-TREE（B+TREE）

- 根据B-Tree的定义，可知检索一次最多需要访问h个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入。（Innodb的数据页是16K，1.2.x支持8K,4K压缩页）。

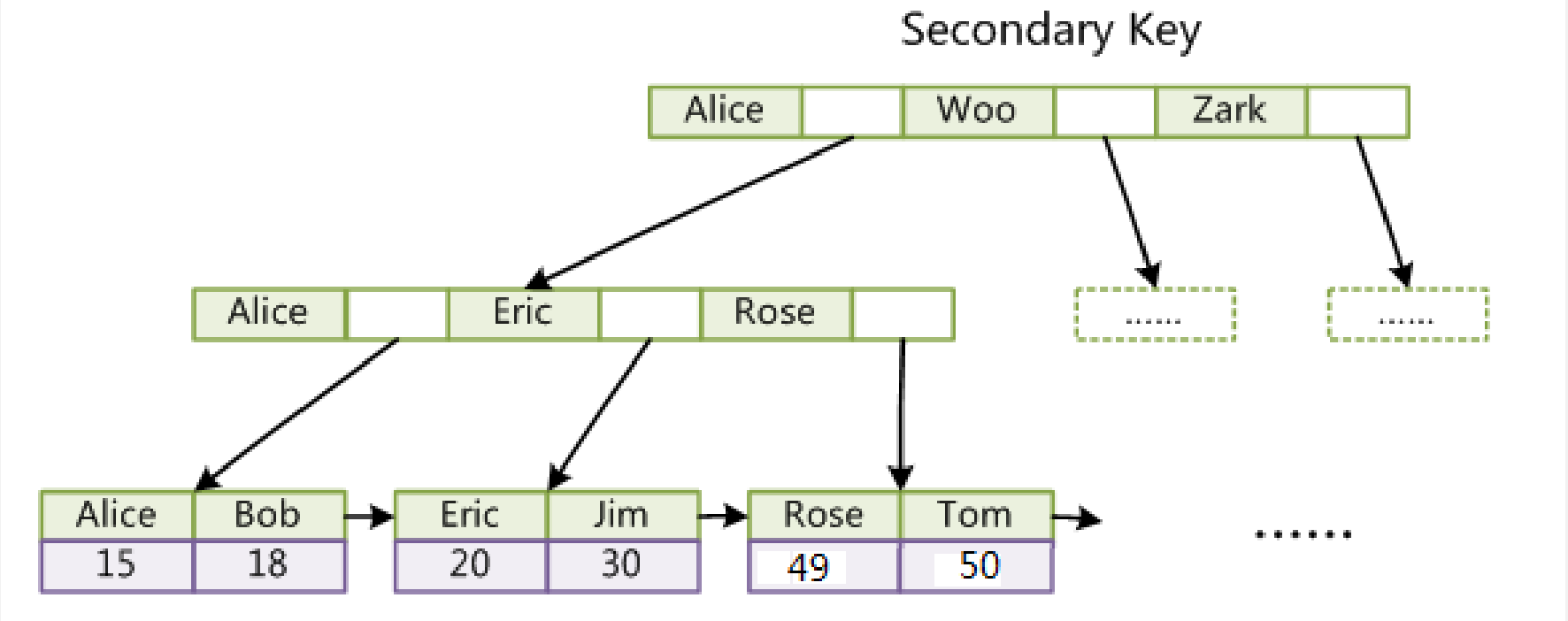
- 每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个node只需一次I/O。

- B-Tree中一次检索最多需要h-1次I/O（根节点常驻内存），渐进复杂度为O(h)=O(logdN)。

Innodb主键索引


Primary Key
15
56
77
15
20
49
.....
.....
15
18
34
77
Bob
Alice
20
30
5
91
Jim
Eric
49
50
22
89
Tom
Rose
.....

Innodb非主键索引



索引遵循的规则

- 短小精悍.
- 整型.
- 避免死索引.
- 避免重复索引.

组合索引

- 创建规则
  - 越小越好
  - 越短越好
- 使用规则
  - 最左策略
  - 全索引扫描

索引存在的问题

- 速度与维护性
  - 查询
  - 更新

- 可选择性
  - 强
  - 弱

```
mysql>  SELECT 1.0/NULLIF(COUNT(DISTINCT(index_field)),0)  FROM table;
```

Cardinality

以总记录数为参考

```
mysql> SELECT TABLES.TABLE_SCHEMA, TABLES.TABLE_NAME,
STATISTICS.INDEX_NAME,
CARDINALITY/TABLE_ROWS AS selective
FROM information_schema.STATISTICS
JOIN information_schema.TABLES
USING (TABLE_SCHEMA, TABLE_NAME)
WHERE non_unique=1 AND TABLE_ROWS/CARDINALITY IS NOT NULL
```

越接近1越好

MySQL 开发规范

开发规范

- 尽量用符合索引利用的规则来写语句
- 避免带有子表的查询
- 不得基于属性函数检索
- 避免全表扫描
- 用UNION ALL
- 用来做JOIN关联的字段类型一定要一致
- 尽量用系统自带函数

开发规范

- 符合每个引擎特性
- 尽量对业务进行批处理
- 谨慎使用触发器
- 谨慎使用视图
- 谨慎使用存储函数
- 适当的使用存储过程
- 适当的使用计划任务

开发规范

- 一份开发规范文档

SQL优化

SQL调优

- EXPLAIN
- SQL读语句优化
- SQL写语句优化

EXPLAIN 可以解决的问题

1. 查看SQL是否正确用到索引
2. 查看SQL用到的索引是否高效
3. 查看是否用到临时表
4. 查看系统优化后的SQL

EXPLAIN的用法

1. Explain [extended]
2. Explain format=json

EXPLAIN例子

```
mysql> EXPLAIN SELECT * FROM tb1 WHERE DATE(log_time) = '2016-02-20'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tb1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 9850
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)
```

EXPLAIN FORMAT例子


```
| {
 "query_block": {
  "select_id": 1,
  "cost_info": {
   "query_cost": "31934.00"
  },
  "table": {
   "table_name": "tb1",
   "access_type": "ALL",
   "rows_examined_per_scan": 157905,
   "rows_produced_per_join": 157905,
   "filtered": "100.00",
```

EXPLAIN FORMAT例子


```
 "cost_info": {
       "read_cost": "353.00",
       "eval_cost": "31581.00",
       "prefix_cost": "31934.00",
       "data_read_per_join": "2M"
     },
"used_columns": [
       "id",
       "log_time"
     ],
     "attached_condition": "(cast(`wochu`.`tb1`.`log_time` as date) = '2016-02-
20')"
   }
  }
} |
```

EXPLAIN 输出

列名	描述
Table	表名
Partition	分区名
select_type	查询类型
Type	关联类型
Key	索引名称
Key_len	索引长度
ref	索引需要引用的列
rows	扫描的行数
filtered	过滤的行数百分比
extra	额外的信息展示

TYPE 列解释

type：这是重要的列，显示优化器正在使用何种类型。

system，符合情况的记录只有一行。const 的一种。

Const, 符合情况的记录最多只有一行。适合主键或者唯一索引的扫描。

Eq_ref, 代表通过JOIN内循环的表扫描次数仅仅为一行。适合主键或者唯一索引的JOIN。

Ref, 非主键之间的JOIN。

Full_text, 全文检索。

Ref_or_null, 和ref类似，但是要扫描null 列。

Index_merge, 多个索引一起用。

Unique_subquery, 替换eq_ref的 子查询。IN (select 主键 from …)

Index_subquery, 替换ref的子查询。IN(select 普通索引列 from …)

Range, 范围扫描。

Index, 扫描整个索引。（相当于小的ALL）

ALL, 全表扫描。

EXPLAIN

- possible_keys：显示可能应用在这张表中的索引。
- key： 实际使用的索引。如果为NULL，则没有使用索引。
- key_len：使用的索引的长度。在不损失精确性的情况下，长度越短越好。
- ref：显示索引的哪一列被使用了，如果可能的话，是一个常数
- rows：MYSQL认为必须检查的用来返回请求数据的行数
- Extra：关于MYSQL如何解析查询的额外信息。这里可以看到的坏的例子是Using temporary和Using filesort，意思MYSQL根本不能使用索引，结果是检索会很慢。

EXPLAIN

- **extra列返回的描述的意义**

- Distinct:一旦MYSQL找到了与行相联合匹配的行，就不再搜索了
- Range checked for each Record（index map:#）:没有找到理想的索引，因此对于从前面表中来的每一个行组合，MYSQL检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一
- Using filesort: 看到这个的时候，查询就需要优化了。MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行
- Using index: 列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候
- Using temporary 看到这个的时候，查询需要优化了。这里，MYSQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上

SQL读语句优化示例

1. 避免函数索引。

SELECT * FROM t WHERE YEAR(d) >= 1994;

SELECT * FROM t WHERE d >= '1994-01-01';

2. 写规范的 JOIN。

SELECT * FROM Country ,CountryLanguage WHERE Country.Code = CountryLanguage.CountryCode;

SELECT * FROM Country JOIN CountryLanguage ON Country.Code = CountryLanguage.CountryCode;

3. 避免数据类型不一致(尤其join)。

SELECT * FROM t WHERE id = 19;

SELECT * FROM t WHERE id = ‘19’;

SQL读语句优化示例

4. 反范式。

SELECT * FROM t WHERE length(column_t) = 5;

SELECT * FROM t WHERE column_length=5;

5. 利用到索引。

SELECT * FROM t WHERE name LIKE '%de%‘

SELECT * FROM t WHERE name LIKE 'de%‘ && SELECT * FROM t WHERE name >= 'de' AND name < 'df‘

6. 读取适当的记录。

SELECT * FROM t WHERE 1;

SELECT * FROM t WHERE 1 LIMIT 10;

7. 读取指定的属性。

SELECT * FROM Country WHERE Name LIKE 'M%';

SELECT Name FROM Country WHERE Name LIKE 'M%';

SQL读语句优化示例

复杂的例子：
1. 不必要的临时表以及不必要的排序。

```
SELECT count(1) AS rs_count FROM
(
SELECT a.id,a.title,a.content,b.log_time,b.name
FROM a,b
WHERE a.content LIKE 'rc_%' AND a.id = b.id
ORDER BY a.title DESC
) AS rs_table;
```

```
SELECT count(1) AS rs_count FROM
(
SELECT a.id FROM a JOIN b
ON a.id = b.id AND a.content LIKE 'rc_%’
) AS rs_table;
```

SQL读语句优化示例

2. 不必要的子表。

```
SELECT * FROM
(
SELECT a.id,a.title,a.content,b.log_time,b.name
FROM a,b
WHERE a.content LIKE 'rc_%' AND a.id = b.id
ORDER BY a.title DESC
) AS rs_table LIMIT 0,30;
```

```
SELECT a.id,a.title,a.content,b.log_time,b.name
FROM a JOIN b
ON a.id = b.id AND a.content LIKE 'rc_%’
ORDER BY a.title DESC LIMIT 0,30;
```

SQL读语句优化示例

3. 缺乏必要的过滤条件。

```
SELECT a.*
FROM a LEFT JOIN b
ON a.id = b.id AND b.name like 'lucy%';
```

```
SELECT a.*
FROM a LEFT JOIN b
ON a.id = b.id AND b.name like ‘lucy%‘ AND a.content like ‘rc_%’;
```

4. 在MySQL 5.5 前对子查询处理不够优化。

```
SELECT count(*) FROM a
WHERE a.id NOT IN (SELECT b.id FROM b);
```

```
SELECT count(a.id)FROM a
LEFT JOIN b USING(id) WHERE b.id IS NULL;
```

SQL读语句优化示例

5. 不同类型的字段做关联查询

```
SELECT a.* FROM a, b WHERE a.id = b.name;
SELECT a.* FROM a, b WHERE a.id = b.id;
```

6. MySQL 对多记录某一点精确定位慢

```
SELECT  * FROM new_ext
ORDER BY id ASC LIMIT 29000,20;

SELECT  * FROM new_ext
WHERE id >= (SELECT id FROM new_ext ORDER BY id ASC LIMIT 29000,1) LIMIT 20 ;
```

（以上由于加快了传送时间，保证了响应时间）

SQL读语句优化示例

7. 对 b 表扫描次数过多，浪费了 CPU 资源原语句：

```
SELECT DISTINCT stop_time,
(SELECT b.tank FROM tb1 b WHERE b.stop_time =a.stop_time AND ne_id =
'1403000000011001') 1403000000011001',
...
(SELECT b.tank FROM tb1 b WHERE b.stop_time =a.stop_time AND ne_id ='1403000000011006')
'1403000000011006 ‘ FROM tb1 a WHERE stop_time>'2010-08-28 00:30:00' AND stop_time<'2010-08-28
01:45:00 ‘ ORDER BY stop_time ASC;
```

注： ne_id,stop_time 是联合主键，stop_time 普通索引

```
SELECT DISTINCT a.stop_time,
CASE WHEN b.ne_id = '1403000000011001' THEN b.tank END AS '1403000000011001',
...
CASE WHEN b.ne_id = '1403000000011006' THEN b.tank END AS '1403000000011006'
FROM tb1 a INNER JOIN tb1 b USING (stop_time)
WHERE a.stop_time>'2010-08-28 00:30:00' AND a.stop_time <'2010-08-28 01:45:00'
ORDER BY a.stop_time ASC;
```

SQL读语句优化示例

8. 随机取记录。

```
SELECT * FROM t1 WHERE 1 ORDER BY RAND() LIMIT 4;
SELECT * FROM t1 WHERE id >= CEIL(RAND()*1000) LIMIT 4;
```

9. 范围查询必须满足最左前缀。

```
SELECT * FROM t1 WHERE id <10 and id > 4 and id1 = 2 ;  index(id,id1)
SELECT * FROM t1 WHERE id1=2 and id<10 and id > 4 ;  index(id1,id)
```

10. 适当的使用 UNION ALL。

```
SELECT * FROM t1 WHERE id =1
UNION
SELECT * FROM t1 WHERE id =2
UNION
SELECT * FROM t1 WHERE id =3;

SELECT * FROM t1 WHERE id =1
UNION ALL
SELECT * FROM t1 WHERE id =2
UNION ALL
SELECT * FROM t1 WHERE id =3;
```

SQL读语句优化示例

11. 强制使用索引 .

select count(*) from t1 where create_time between ‘2016-06-01 20:00:00’ and ‘2016-06-01 21:00:00’ order by id asc limit 50;

select count(*) from t1 force index(index_create_time) where create_time between ‘2016-06-01 20:00:00’ and ‘2016-06-01 21:00:00’ order by id asc limit 50;

12. 避免使用 HAVING 子句。

```
SELECT * FROM
(
SELECT COUNT(1),id2 FROM t1 WHERE 1 GROUP BY id2 HAVING id2 >50
) T LIMIT 10;
```

```
SELECT * FROM
(
SELECT COUNT(1),id2 FROM t1 WHERE id2 > 50 GROUP BY id2
) T LIMIT 10;
```

13. 显示指定列属于那张表。

SELECT id2 FROM t1 STRAIGHT_JOIN t2 WHERE t1. id = t2.id;

SELECT t1.id2 FROM t1 STRAIGHT_JOIN t2 WHERE t1.id = t2.id;

SQL读语句优化示例

14. 避免在列上进行计算。

```
SELECT id2 FROM t1 WHERE id2*10 > 100;
SELECT id2 FROM t1 WHERE id2 > 100 DIV 10;
```

15. 避免不确定的范围扫描。

```
select id2 from t1 where id2 > 10 limit 10;
select id2 from t1 where id2 >= 11 limit 10;
```

16. 避免不等于。

```
SELECT COUNT(1) FROM t3 WHERE id2 <> 417;

SELECT(
(SELECT COUNT(*) FROM t3 WHERE 1) - (SELECT COUNT(id2) FROM t3 WHERE id2 =
417)
) AS t;
```

17. 状态值加索引。

```
SELECT * FROM t_status WHERE `status` = 1;
alter table t_status add key idx_status (`status`);
```

SQL读语句优化示例

18. 基于场景优化。

```
select * from t where t_stats=1 and t_type in (0,1);
566 rows in set (5.12 sec)
```

```
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+---------------+
| id | select_type | table | type | possible_keys |  key     | key_len|    ref     | rows       | Extra           |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE     | t       | ALL | NULL           | NULL | NULL   | NULL| 5000000 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+---------------+
```

SQL读语句优化示例

18. 基于场景优化。

```
select count(*),t_stats from t group by t_stats;
+----------+----------------+
| count(*) |    t_stats     |
+----------+----------------+
|     1000 |             -1 |
|  2099000 |              0 |
|   900000 |              1 |
+----------+----------------+
select count(*),t_type from t group by t_type;
+----------+------------+
| count(*) |    t_type  |
+----------+------------+
|     2000 |          0 |
|  2000000 |          1 |
|  2998000 |          3 |
+----------+------------+
```

```
alter table t add index idx_stats_type(t_stats,t_type);

select * from t where t_stats=1 and t_type in (0,1);
566 rows in set (0.11 sec)
```

SQL读语句优化示例

19. 避免无关联的SQL连接。

select t1.id,t2.name from t1,t2,t3 where t1.id=t2.id and t2.age=1;

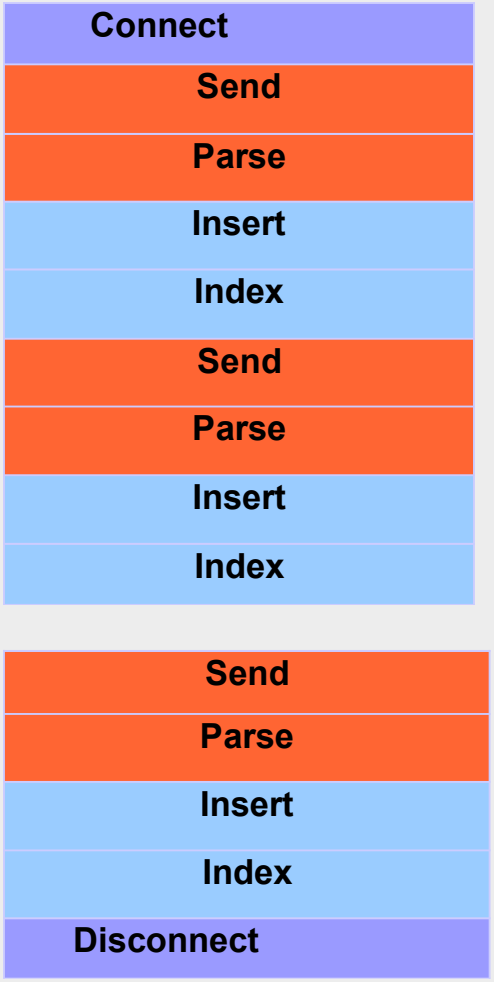
select t1.id,t2.name from t1,t2 where t1.id=t2.id and t2.age=1;

SQL写语句优化

多VALUES插入

```
mysql> INSERT INTO ID_table (ID,Name) VALUES

     (1,'Max'),(2,'Pat'),(3,'Roland'),(4,'Jeff'),(5,'S
  arah');
```



SQL写语句优化

INSERT … SELECT

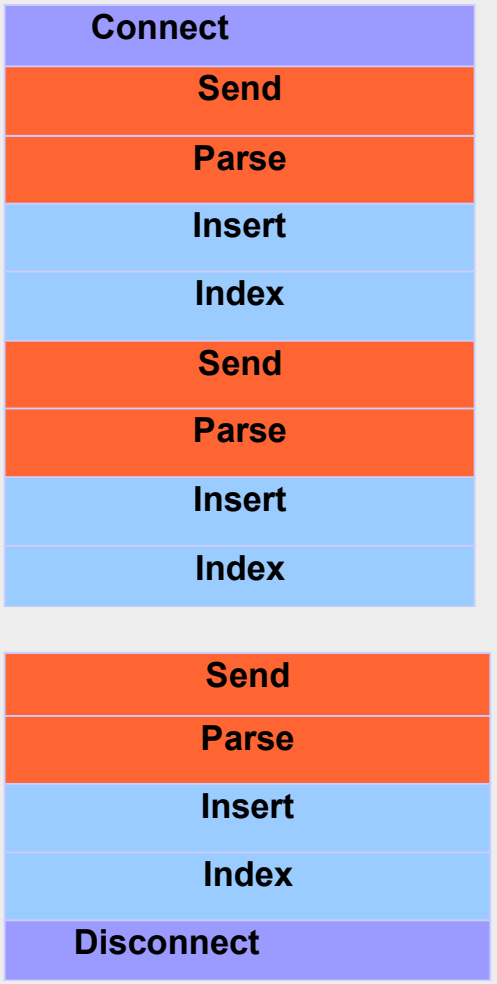
```
mysql> truncate table city_bak;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into city_bak select * from city;
Query OK, 4079 rows affected, 1 warning (0.11 sec)
Records: 4079  Duplicates: 0  Warnings: 1
```

REPLACE… SELECT

```
mysql> replace into city_bak select * from city;
Query OK, 8158 rows affected, 2 warnings (0.05 sec)
Records: 4079  Duplicates: 4079  Warnings: 2

mysql> select count(*) from city_bak;
+----------+
| count(*) |
+----------+
|     4079 |
+----------+
1 row in set (0.00 sec)
```



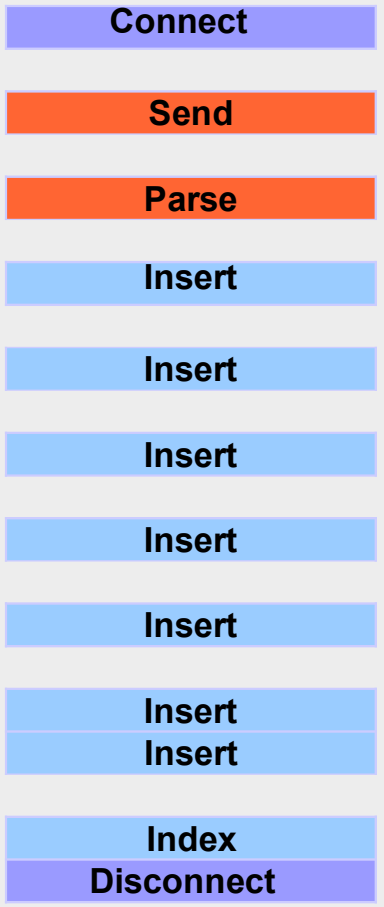
SQL写语句优化

LOAD DATA … INFILE

```
mysql> select * from city into outfile 'j:/city.txt';
Query OK, 4079 rows affected (0.04 sec)

mysql> create table city_bak like city;
Query OK, 0 rows affected (0.02 sec)

mysql> load data infile 'j:/city.txt' into table city_bak;
Query OK, 4079 rows affected, 823 warnings (0.06 sec)
Records: 4079  Deleted: 0  Skipped: 0  Warnings: 823
```



SQL写语句优化

用事务来插入

```
mysql> START TRANSACTION;
mysql> INSERT INTO city (NULL,'Bryson City','USA','North Carolina', 7800);
mysql> INSERT INTO city (NULL,'Virginia City','USA','Virginia',5467);
mysql> INSERT INTO city (NULL,'Morgantown','USA','West Virginia',11435);
mysql> INSERT INTO city (NULL,'Julian','USA','Pennsylvania',4356);
mysql> COMMIT;
```

SQL写语句优化

提升InnoDB 写入

禁止自动提交

```
mysql> SET AUTOCOMMIT=0;
... SQL import statements ...
mysql> COMMIT;
```

禁止唯一性检查

```
mysql> SET UNIQUE_CHECKS=0;
... import operation ...
mysql> SET UNIQUE_CHECKS=1;
```

禁止外键完整性检查

```
mysql> SET FOREIGN_KEY_CHECKS=0;
... import operation ...
mysql> SET FOREIGN_KEY_CHECKS=1;
```

SQL审核

自动化运维平台

SQL审核：
(1)语法检查
(2)语义检查
(3)基于规则的SQL审核
http://wiki.jdb-dev.com/pages/viewpage.action?pageId=9051400

(4)完善基于规则SQL审核(研发中)
(5)事前/事中/事后自动化审核(研发中)

谢谢！