

MySQL培训之SQL开发优化指导

目录

SQL的执行原理

SQL的基本语法

SQL的开发规范

查询优化详解

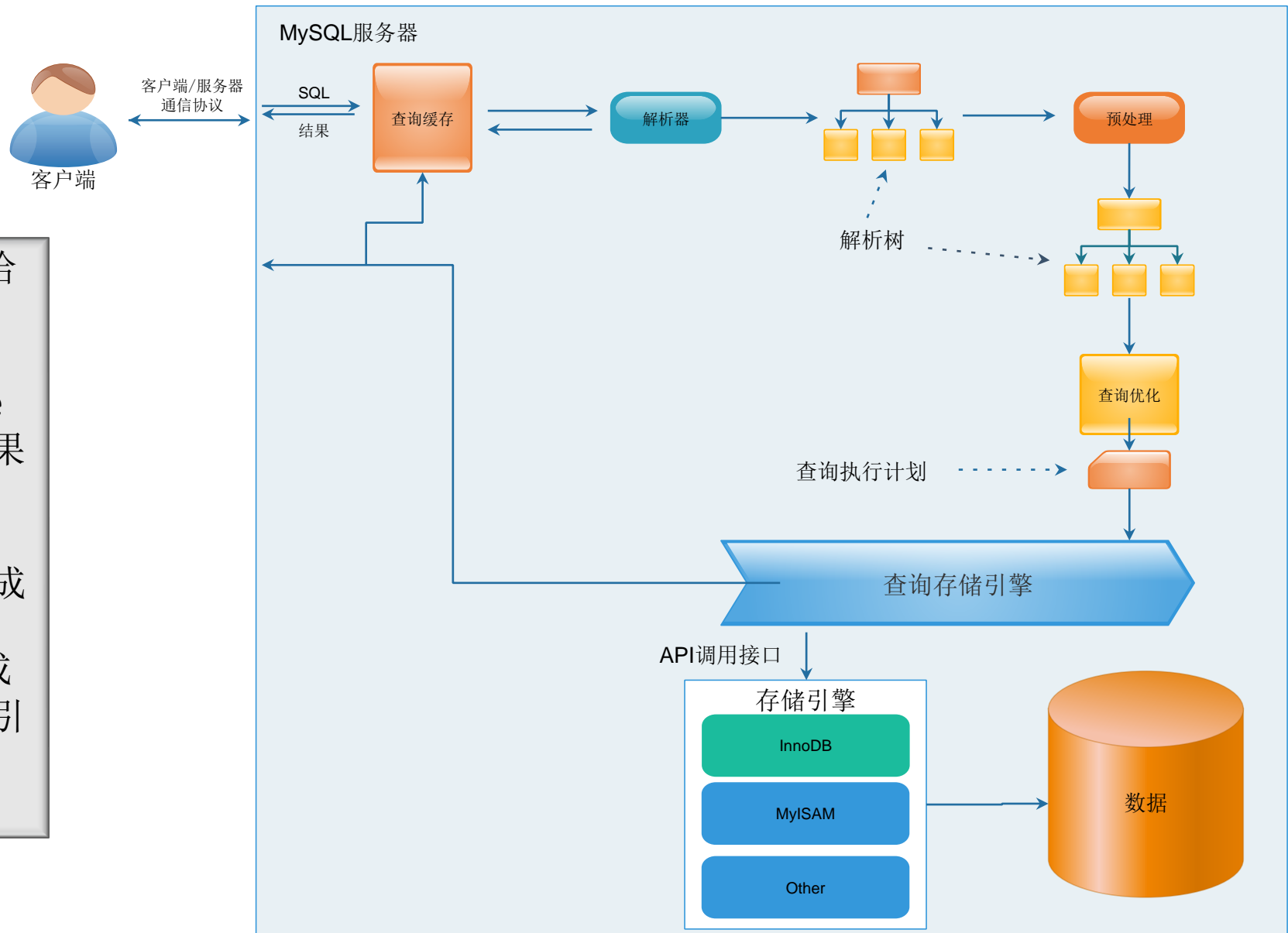
优化案例分享

SQL优化总结

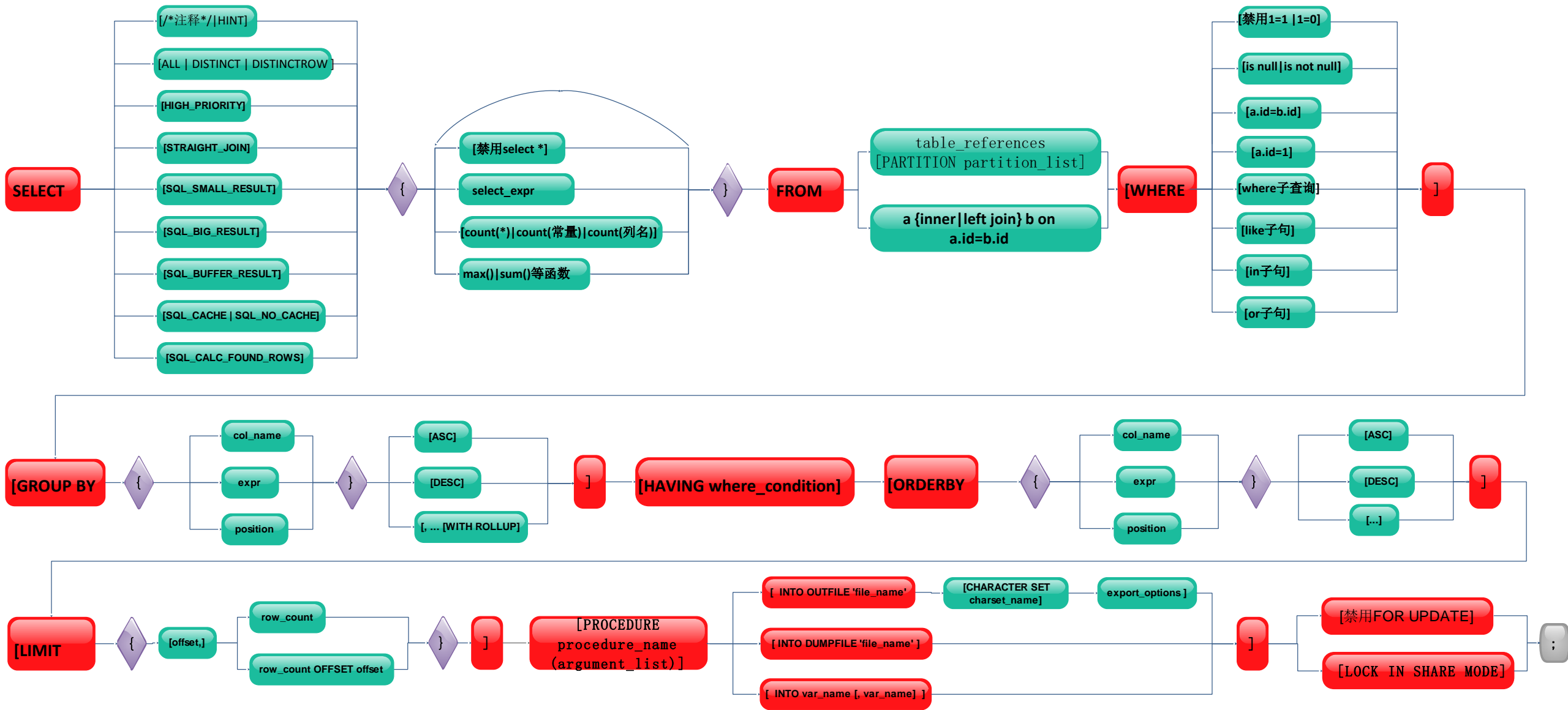
Select是如何执行的？

MySQL select是如何执行的

- 1.客户端发送一条查询给服务器。
- 2.服务器检查查询缓存，如果命中了缓存，则like返回存储在缓存中的结果。否则进入下一阶段。
- 3.服务器进行SQL解析、预处理，再由优化器生成对应的执行计划。
- 4.MySQL根据优化器生成的执行计划，调用存储引擎的API来执行查询。
- 5.将结果返回给客户端。



Select的基本语法



SQL的开发规范原则

【规则】SQL统一小写，词间一空格

`select name from tabA where name='bon';`

`select Name from tabA where name='bon';`（大小写不一致）

`select name from tabA where name='bon';`（空格不一致）

【禁止】明确查询字段，禁用select *

- 1.防止表结构变更导致程序异常
- 2.明确字段，减少递归数据字典访问
- 3.会导致无法使用覆盖索引，增加访问成本。

【规则】禁止在开发过程中的SQL语句中使用hint

- 1.应用程序中SQL部分禁止使用Hint，特殊情况需提供原因
- 2.不使用数据库特有的属性，规范程序开发，减少特定数据源依赖

【STRAIGHT_JOIN仅限优化使用，开发禁用】：

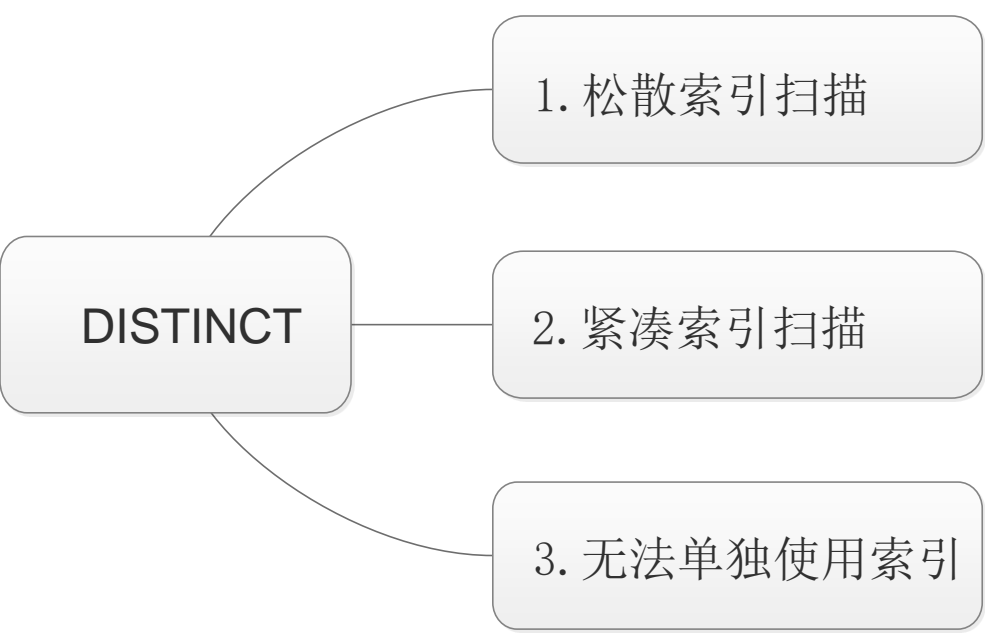
用于调整表关联的驱动顺序，可用于测试，开发禁用



查询的优化——DISTINCT

语法: **SELECT** → **[ALL | DISTINCT | DISTINCTROW]** → **select_expr** → **FROM**

[ALL]: SELECT语句(默认值)返回所有的行, 包括重复值。
[DISTINCT]:与DISTINCTROW同义, 指定去除结果集中的重复值, 不可与ALL同用。其实是在GROUP BY 之后的每组中只取出一条记录, 但是不需要排序。



```
mysql> explain select distinct dept_no from dept_emp;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | dept_emp | NULL | range | PRIMARY,dept_no | dept_no | 12 | NULL | 9 | 100.00 | Using index for group-by |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
mysql> explain select distinct emp_no from dept_emp where dept_no='d004';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | dept_emp | NULL | ref | PRIMARY,dept_no | dept_no | 12 | const | 128052 | 100.00 | Using where; Using index |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
mysql> EXPLAIN SELECT DISTINCT emp_no FROM dept_emp WHERE dept_no > 'd004' AND dept_no < 'd006';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | ... | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | dept_emp | ... | range | PRIMARY,dept_no | dept_no | 12 | NULL | 148040 | 100.00 | Using where; Using index; Using temporary |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

NOTE: DISTINCT关键在于利用好索引, 无法利用索引的时候不要在大结果集上进行distinct操作。

查询的优化——COUNT()



```
root(ngmm_dev) >explain SELECT COUNT(*) FROM t_mm_lbpic;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL | Select tables optimized away |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

查询优化器阶段自动优化

```
root(ngmm_dev) >explain SELECT COUNT(*) FROM t_mm_lbpic WHERE prov_code IN ('000','100','200');
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t_mm_lbpic | NULL | ALL | NULL | NULL | NULL | NULL | 96 | 30.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

ALL全表扫描

创建索引: create index i_t_mm_lbpic_test1 on t_mm_lbpic(prov_code);


```
root(ngmm_dev) >explain SELECT COUNT(*) FROM t_mm_lbpic WHERE prov_code IN ('000','100','200');
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | t_mm_lbpic | NULL | range | i_t_mm_lbpic_test1 | i_t_mm_lbpic_test1 | 33 | NULL | 9 | 100.00 | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

使用索引RANGE

NOTE: 查询表的行数时, 使用count(*)替代count(列名)或count(常量)

查询的优化——FROM/表关联

- 【强制】**禁止三个表以上**的关联，多表关联可程序侧拆分
数据字典类型表在程序启动时缓存、通过redis等NoSQL缓存字典类信息，减少多表关联
- 【强制】**禁止跨库**操作（比如：select、update等）：禁止所有跨库操作，模型设计初期应规范
- 【规则】From列表中，常数表优先，字典表或小表其次，**大表最后**
- 【原则】ON、USING子句中的列确认有索引。一般只需**被驱动表的关联列**上有**索引**即可。
- 【原则】最好是能转化为**INNER JOIN**，LEFT JOIN的成本比INNER JOIN高很多。
- 【建议】多表连接时，使用表的**别名**来引用列



表关联优化建议

- 确保 ON 或者 USING 子句中的列上有索引。在创建索引的时候就要考虑到关联的顺序。当表 A 和表 B 用列 c 关联的时候，如果优化器的关联顺序是 B、A，那么就不需要在 B 表的对应列上建上索引。没有用到的索引只会带来额外的负担。一般来说，除非有其他理由，否则只需要在关联顺序中的第二个表的相应列上创建索引。
- 确保任何的 GROUP BY 和 ORDER BY 中的表达式只涉及到一个表中的列，这样 MySQL 才有可能使用索引来优化这个过程。
- 当升级 MySQL 的时候需要注意：关联语法、运算符优先级等其他可能会发生变化的地方。因为以前是普通关联的地方可能会变成笛卡儿积，不同类型的关联可能会生成不同的结果等。

查询的优化——WHERE条件及关联隐式转换

【强制】禁止隐式转换，保持变量类型与字段类型一致

```
SQL1:
select emp_no,from_date from dept_emp1
where dept_no='404838';

SQL2:
select emp_no,from_date from dept_emp1
where dept_no=404838;
```

```
CREATE TABLE `dept_emp1` (
  `emp_no` int(11) NOT NULL,
  `dept_no` char(4) NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date NOT NULL,
  PRIMARY KEY (`emp_no`,`dept_no`),
  KEY `dept_no` (`dept_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```



```
>explain select emp_no,from_date from dept_emp1 where dept_no='404838';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | dept_emp1 | NULL | ref | dept_no | dept_no | 12 | const | 1 | 100.00 | Using index condition |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

>explain select emp_no,from_date from dept_emp1 where dept_no=404838;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | dept_emp1 | NULL | ref | dept_no | dept_no | 12 | const | 1 | 10.00 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 3 warnings (0.00 sec)

>show warnings;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Level | Code | Message |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Warning | 1739 | Cannot use ref access on index 'dept_no' due to type or collation conversion on field 'dept_no' |
| Warning | 1739 | Cannot use range access on index 'dept_no' due to type or collation conversion on field 'dept_no' |
| Note | 1003 | /* select#1 */ select `ngwf_dev`.`dept_emp1`.`emp_no` AS `emp_no`,`ngwf_dev`.`dept_emp1`.`from_date` AS `from_date` from `ngwf_dev`.`dept_emp1` where `dept_no`=404838 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

【强制】多表关联所涉及的字段必须有索引
关联字段上必须有索引，并且类型一致。否则会不走索引
【规则】禁止对“=”左侧字段使用函数、运算
“=”号左侧通常为字段，对字段上使用函数，会导致索引失效。
MySQL无目前基于函数索引。

隐式转换影响：

1.性能问题
当列的数据类型为字符型时，可能会隐式转换导致不能走索引情况。问题不可控，从而引发数据库性能问题

2.安全问题
假如 password 类型为字符串，查询条件为 int 0 则会匹配上。

查询的优化——WHERE处理NULL值

【禁止】禁止使用 not、!=、<>、is null, is not null 条件

MySQL处理null值:

- 1.使用DISTINCT、GROUP BY或ORDER BY时，所有NULL值将被视为是等同的。
- 2.使用ORDER BY时，首先将显示NULL值，如果指定了DESC按降序排列，那么NULL值将在最后面显示
- 3.对于聚合（累计）函数，如COUNT()、MIN()和SUM()，将忽略NULL值，COUNT(*)，计数行而不是单独的列值。

特殊情况1：如果将NULL值插入TIMESTAMP列，那么将插入当前日期和时间。

特殊情况1：如果将NULL值插入具有AUTO_INCREMENT属性的整数列，那么将插入序列中的下一个编号。

(1)使用CASE语句

```
SELECT CASE
    WHEN id IS NULL THEN 0
    ELSE id
END
FROM test2;
```

(2)使用COALESCE函数

```
SELECT COALESCE( id , 0 ) FROM test2;
```

COALESCE(value,...)函数：返回值为列表当中的第一个非NULL值，在没有非NULL值的情况下返回值将为NULL。

(3)使用IFNULL函数

```
SELECT ifnull(id,0) FROM test2;
```

IFNULL(expr1, expr2)函数：

假如expr1不为NULL，则IFNULL()的返回值为expr1；否则其返回值为expr2。IFNULL()的返回值是数字还是字符串取决于其所使用的语境。

(4)使用IF函数

```
SELECT IF (id is null, 0, id) AS id FROM test2;
```

IF(expr1,expr2,expr3):如果expr1是TRUE，则IF()的返回值为expr2；否则返回值为expr3。IF()的返回值是数字还是字符串视其所在的语境而定

查询的优化—WHERE子查询

【规则】避免使用子查询、or，将子查询转化为表连接方式，or转化为in

优化子查询：

1.使用连接方式改写子查询

示例1: `SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (SELECT column1 FROM t2);`

改写: `SELECT DISTINCT t1.column1 FROM t1, t2 WHERE t1.column1 = t2.column1;`

示例2: `SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);`

改写: `SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);`

还可以改写成如下LEFT JOIN:

`SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id WHERE table2.id IS NULL;`

2.把子句从子查询的外部转移到内部

示例1: `SELECT * FROM t1 WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);`

改写: `SELECT * FROM t1 WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);`

示例2: `SELECT (SELECT column1 FROM t1) + 5 FROM t2;`

改写: `SELECT (SELECT column1 + 5 FROM t1) FROM t2;`

示例3: `SELECT * FROM t1`

`WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1 AND t2.column2=t1.column2);`

改写: `SELECT * FROM t1 WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);`

3.对于只返回一行的无关联子查询用‘=’代替‘in’

示例1: `SELECT * FROM t1 WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);`

改写: `SELECT * FROM t1 WHERE t1.col_name= (SELECT a FROM t2 WHERE b = some_const);`

对于数据库来说，在绝大部分情况下，连接会比子查询更快。使用连接的方式，MySQL优化器一般可以生成更佳的执行计划，可以预先装载数据，更高效地处理查询。而子查询往往需要运行重复的查询，子查询生成的临时表上也没有索引，因此效率会更低。

查询的优化——WHERE OR子句

【规则】避免使用子查询、or，将子查询转化为表连接方式，or转化为in

优化or子句：

1.or子句全部相同，则改为in

示例：select * from t1 where a=1 or a=3;

改为：select * from t1 where a in(1,3);

2.or子句具有公共子序列前缀的，请在or公共部分建立索引

示例：（如下需要在a列上创建索引）

select * from t1 where (a=1 and b=2) or (a=3 and c=4);

3.若无公共，则建议改为union all，并为每部分建立索引

示例 select * from t1 where a=1 or b=2;

转换：select * from t1 where a=1

union all

select * from t1 where b=2;

4.复杂or条件，编写后进行推理改写

示例：

4.1) ((a AND b) AND c OR (((a AND b) AND (c AND d))))

调整为->>> (a AND b AND c) OR (a AND b AND c AND d)

4.2)(a<b AND b=c) AND a=5

调整为->>> b>5 AND b=c AND a=5

4.3) (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)

调整为->>> B=5 OR B=6

再调整为->>> B in (5,6)

查询的优化——WHERE条件不走索引



如何创建合适的索引?

where <col> !=

连接索引列运算或使用函数

where <col> <>

MySQL本身的问题或其他

单列索引 or 复合索引

不会走索引的where条件

like查询或者_开头

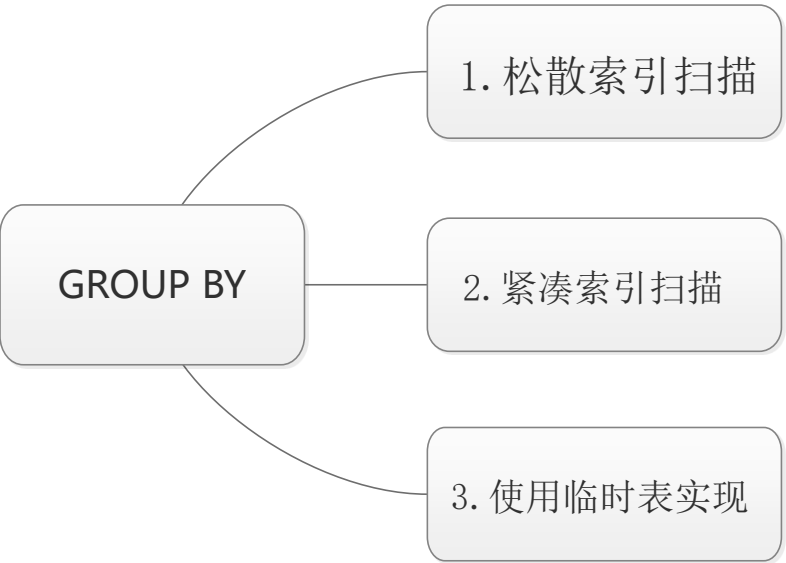
数据类型不匹配

复合索引未使用前导列

where ... in, not exist 等等

查询的优化——GROUP BY 分类

分为3种情况：



索引：KEY `idx_gid_uid_gc` (`group_id`,`user_id`,`gmt_create`)

```
示例: mysql> explain SELECT user_id,max(gmt_create) FROM group_message WHERE group_id < 10 GROUP BY user_id;
mysql> explain SELECT user_id,max(gmt_create) FROM group_message WHERE group_id < 10 GROUP BY user_id;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | ... | type | possible_keys | key | ... | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | group_message | ... | range | idx_gid_uid_gc | idx_gid_uid_gc | ... | Using where; Using index; Using temporary; Using filesort |
+----+-----+-----+-----+-----+-----+-----+-----+
优化:
mysql> explain SELECT user_id,max(gmt_create) FROM group_message WHERE group_id < 10 GROUP BY group_id,user_id;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | ... | type | possible_keys | key | ... | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | group_message | ... | range | idx_gid_uid_gc | idx_gid_uid_gc | ... | Using where; Using index |
+----+-----+-----+-----+-----+-----+-----+-----+
```

优化：group by增加索引前缀列group_id。使其满足“必须在同一个索引中最前面的连续位置”

```
mysql> EXPLAIN SELECT max(gmt_create) FROM group_message WHERE group_id = 2 GROUP BY user_id;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | ... | type | possible_keys | key | ... | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | group_message | ... | ref | idx_gid_uid_gc | idx_gid_uid_gc | ... | 100.00 | Using where; Using index |
+----+-----+-----+-----+-----+-----+-----+-----+
```

分析：group by前缀列group_id为常量，可以省略group by中该列，只需user_id即可。

```
mysql> mysql> EXPLAIN SELECT max(gmt_create) FROM group_message WHERE group_id > 1 and group_id < 10 GROUP BY user_id;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | ... | type | possible_keys | key | ... | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | group_message | ... | range | idx_gid_uid_gc | idx_gid_uid_gc | ... | Using where; Using index; Using temporary; Using filesort |
+----+-----+-----+-----+-----+-----+-----+-----+
```

分析：where条件中group_id > 1 and group_id < 10给的范围或者in条件，group_id 并不是一个常量条件，而是一个范围，而且 GROUP BY字段为 user_id。

查询的优化——GROUP BY优化思路及策略

优化思路

1)尽可能**利用索引**来完成 GROUP BY 操作，当然最好是**松散索引扫描**（range）的方式最佳。

前提条件：

(1).GROUP BY 条件字段必须在同一个索引中最前面的连续位置

(2).使用GROUP BY 的同时，只能使用 MAX 和 MIN 这两个聚合函数

(3).如果引用到了该索引中 GROUP BY 条件之外的字段条件的时候，必须以**常量形式存在**。

在系统允许的情况下，我们可以通过调整索引或者调整 Query 这两种方式来达到目的；

2) 当无法使用索引完成 GROUP BY 的时候，由于要使用到临时表且需要 filesort，所以我们必须要有足够的 **sort_buffer_size** 来供 MySQL 排序的时候使用，而且尽量**不要进行大结果集的 GROUP BY** 操作，因为如果超出系统设置的临时表大小的时候会出现将临时表数据 copy 到磁盘上面再进行操作，这时候的排序分组操作性能将是成数量级的下降

3)**小技巧**，在**无排序需求**的group by操作中在整个语句最后添加一个以 null 排序（**ORDER BY null**）的子句，可以避免在无法利用索引的情况下的filesort操作。

优化策略

1.Group by字段建立索引

2.修改业务逻辑，减少大结果集的group by

3.多表连接后的group by，使用第一个表（驱动表）上的列

4.group by增加 order by null避免不必排序

5.使用非GROUP BY的列来代替GROUP BY的列（如果分组结果是相同的）

6.可以考虑使用Sphinx等产品来优化GROUP BY语句



查询的优化——ORDER BY索引排序

【索引排序】排序使用索引提高效率，减少文件排序

- 1.order by顺序和索引顺序一致，且所有列一致升序或降序
- 2.当多表连接时，order by所有列使用第一个表，此时同样遵循最左前缀法则
- 3.order by字段是索引最左连续字段
- 4.如果有 order by 的场景，请注意利用索引的有序性。order by 最后的字段是组合索引的一部分，正例：并且放在索引组合顺序的最后，避免出现 file_sort 的情况，影响查询性能。

where a=? and b=? order by c; 索引：a_b_c

反例：

索引中有范围查找，那么索引有序性无法利用，

如：WHERE a>10 ORDER BY b; 索引 a_b无法排序。

- 5.where + order by可利用复合索引实现order by优化。

```
SELECT [column1],[column2],... FROM [TABLE] WHERE [columnX] = [value] ORDER BY [sort];
```

反例：

```
SELECT [column1],[column2],... FROM [TABLE] WHERE [columnX] IN ([value1],[value2],...) ORDER BY[sort];
```

优化：in 改写为对应的union all语句

```
SELECT [column1],[column2],... FROM [TABLE]
```

```
WHERE [columnX] =[value1] ORDER BY[sort];
```

union all

```
SELECT [column1],[column2],... FROM [TABLE]
```

```
WHERE [columnX] =[value2] ORDER BY[sort];
```


查询的优化——ORDER BY无法索引排序

```
SELECT a.id
      ,a.serviceTreeId
      ,a.staffId
      ,a.sequence
      ,a.proviceId
      ,b.NAME
      ,b.nodedesc
FROM t_mm_personalservicetree a
LEFT JOIN t_mm_servicetree b ON a.serviceTreeId = b.id
WHERE a.staffId = 'cs110'
      AND a.proviceId IN (
          '00040008'
          , '00030009'
          , '00030007')
ORDER BY a.sequence;
```

a.proviceId in 调整为a.proviceId =

```
SELECT a.id
      ,a.serviceTreeId
      ,a.staffId
      ,a.sequence
      ,a.proviceId
      ,b.NAME
      ,b.nodedesc
FROM t_mm_personalservicetree a
LEFT JOIN t_mm_servicetree b ON a.serviceTreeId = b.id
WHERE a.staffId = 'cs110'
      AND a.proviceId = '00040008'
ORDER BY a.sequence;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	ALL	NULL	NULL	NULL	NULL	95	3.00	Using where; Using filesort
1	SIMPLE	b	NULL	eq_ref	PRIMARY	PRIMARY	182	ngmm_dev.a.SERVICETREEID	1	100.00	Using where

create index i_test1 on t_mm_personalservicetree (staffId,proviceId,sequence);

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	range	i_test1	i_test1	245	NULL	3	100.00	Using index condition; Using filesort
1	SIMPLE	b	NULL	eq_ref	PRIMARY	PRIMARY	182	ngmm_dev.a.SERVICETREEID	1	100.00	Using where

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	a	NULL	ref	i_test1	i_test1	245	const,const	1	100.00	Using index condition
1	SIMPLE	b	NULL	eq_ref	PRIMARY	PRIMARY	182	ngmm_dev.a.SERVICETREEID	1	100.00	Using where

建议: order by尽量使用索引排序, in能避免则避免, 可考虑union all避免filesort, 也可改为多个等值后在应用层实现连接。另禁止使用关键字作列名。

查询的优化——LIMIT偏移量过大

【规则】禁止分页查询偏移量过大，如limit 10000,10

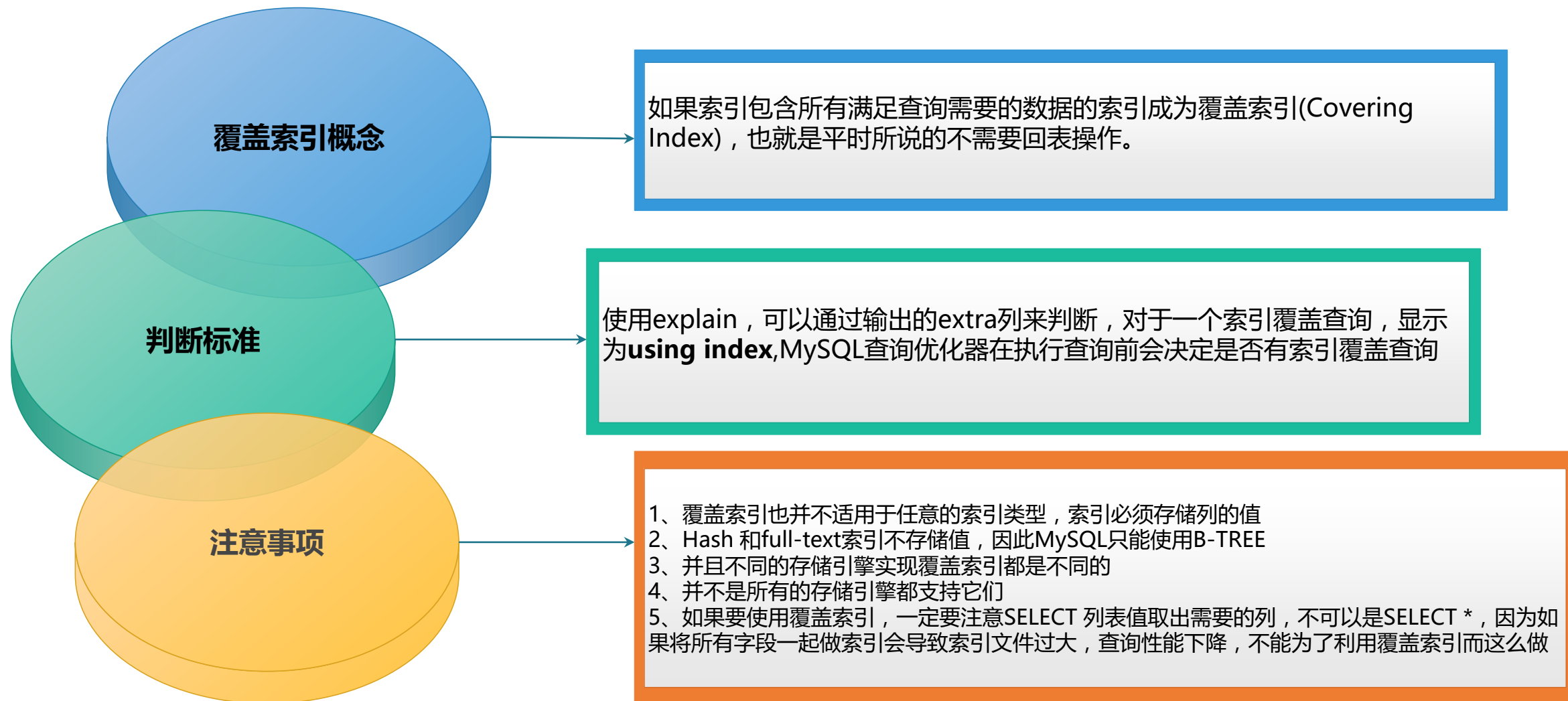
```
CREATE TABLE `employees` (  
  `emp_no` int(11) NOT NULL,  
  `birth_date` date NOT NULL,  
  `first_name` varchar(14) NOT NULL,  
  `last_name` varchar(16) NOT NULL,  
  `gender` enum('M','F') NOT NULL,  
  `hire_date` date NOT NULL,  
  PRIMARY KEY (`emp_no`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8  
root(employees) >show profiles;
```

Q_ID	Duration	Query
1	0.09779125	select birth_date from employees where gender='M' order by hire_date limit 10
2	1.06902425	create index i_employees_gender_hire on employees(gender,hire_date)
3	0.00031550	select birth_date from employees where gender='M' order by hire_date limit 10
4	0.01447400	select birth_date from employees where gender='M' order by hire_date limit 10000,10
5	0.13611950	select birth_date from employees where gender='M' order by hire_date limit 100000,10
6	0.02908750	SELECT birth_date FROM employees INNER JOIN (SELECT emp_no FROM employees WHERE gender = 'M' ORDER BY hire_date limit 100000,10) AS x using (emp_no)
7	0.03081975	SELECT birth_date FROM employees INNER JOIN (SELECT emp_no FROM employees WHERE gender = 'M' ORDER BY hire_date limit 110000,10) AS x using (emp_no)
8	0.15999875	select birth_date from employees where gender='M' order by hire_date limit 110000,10

总结

1. 避免设置offset值，也就是避免丢弃记录。
2. 尽量使用索引排序完成件事文件排序。
3. 最好的解决方式限制用户翻页。

查询的优化——覆盖索引



案例分享——到底用了索引的哪些列

```
>show create table dept_emp\G;
***** 1. row *****
      Table: dept_emp
Create Table: CREATE TABLE `dept_emp` (
  `emp_no` int(11) NOT NULL,
  `dept_no` char(4) NOT NULL,
  `from_date` date NOT NULL,
  `to_date` date NOT NULL,
  PRIMARY KEY (`emp_no`,`dept_no`),
  KEY `dept_no` (`dept_no`),
  CONSTRAINT `dept_emp_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES `employees` (`emp_no`) ON DELETE CASCADE,
  CONSTRAINT `dept_emp_ibfk_2` FOREIGN KEY (`dept_no`) REFERENCES `departments` (`dept_no`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

SQL: select emp_no from dept_emp where dept_no='d001';

```
>explain select emp_no from dept_emp where dept_no='d001';
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | dept_emp | NULL | ref | dept_no | dept_no | 12 | const | 38276 | 100.00 | Using index |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

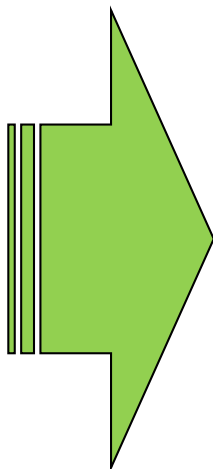
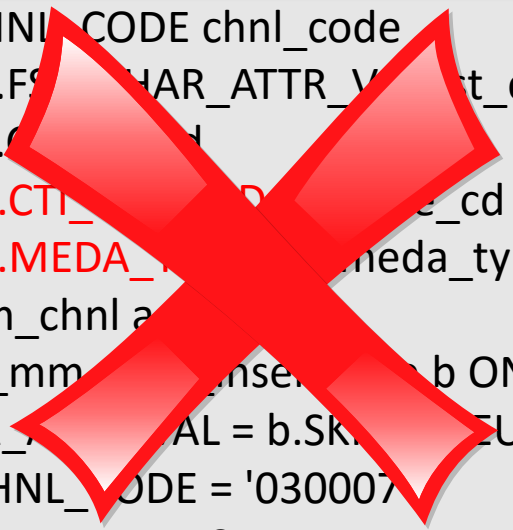
```
>explain select name from test_len where c1=1 and c2=2 order by c3;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | test_len | NULL | ref | i_test_len_c | i_test_len_c | 8 | const,const | 1 | 100.00 | Using index condition |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```



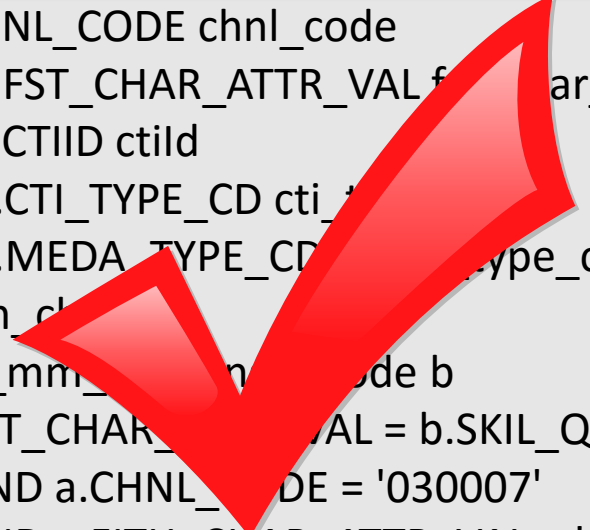
案例分享——SQL改写1

关键字: left join → join

```
SELECT a.CHNL_CODE chnl_code
      ,a.FST_CHAR_ATTR_VAL fst_char_attr_val
      ,a.CTIID ctild
      ,b.CTI_TYPE_CD cti_type_cd
      ,b.MEDA_TYPE_CD meda_type_cd
FROM t_mm_chnl a
LEFT JOIN t_mm_chnl_insertcode b ON
a.FST_CHAR_ATTR_VAL = b.SKIL_QUEUE_ID
WHERE a.CHNL_CODE = '030007'
      AND a.FITH_CHAR_ATTR_VAL = '000';
```



```
SELECT a.CHNL_CODE chnl_code
      ,a.FST_CHAR_ATTR_VAL fst_char_attr_val
      ,a.CTIID ctild
      ,b.CTI_TYPE_CD cti_type_cd
      ,b.MEDA_TYPE_CD meda_type_cd
FROM t_mm_chnl a
      JOIN t_mm_chnl_insertcode b ON
a.FST_CHAR_ATTR_VAL = b.SKIL_QUEUE_ID
WHERE a.CHNL_CODE = '030007'
      AND a.FITH_CHAR_ATTR_VAL = '000';
```



```
>show create table t_mm_chnl_insertcode\G
      Table: t_mm_chnl_insertcode
Create Table: CREATE TABLE `t_mm_chnl_insertcode` (
      .....
      `CTI_TYPE_CD` varchar(100) NOT NULL COMMENT 'CTI类型代码',
      `MEDA_TYPE_CD` varchar(100) NOT NULL COMMENT '媒体类型代码',
      .....
      PRIMARY KEY (`SKIL_QUEUE_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='技能队列'
```

【原则】最好是能转化为INNER JOIN，**LEFT JOIN的成本**比INNER JOIN高很多。

LEFT JOIN 改为join :b的select的列CTI_TYPE_CD，MEDA_TYPE_CD都是not null，所以无需left join。或者where条件有后边的表是非空条件(除is null外)，则等同于JOIN

案例分享——SQL改写2

```
查询黑名单池状态 t_mm_blacklist
SELECT t.SERIALNO
      ,t.USER_ID
      ,t.CHNL_CODE
      ,t.CHNL_NM
      ,t.USER_TYPE_CD AS USER_CATAGORY
      ,t.CNTMNG_SWFTNO
      ,t.BLST_TIME
FROM t_mm_blacklist t
WHERE t.USER_TYPE_CD = '0'
      AND t.USER_ID = '15485599967'
UNION
SELECT t.SERIALNO
      ,t.USER_ID
      ,t.CHNL_CODE
      ,t.CHNL_NM
      ,t.USER_TYPE_CD AS USER_CATAGORY
      ,t.CNTMNG_SWFTNO
      ,t.BLST_TIME
FROM t_mm_blacklist t
WHERE t.USER_ID = '15485599967'
      AND t.USER_TYPE_CD = '1'
      AND t.BLST_TIME > '2017-05-16 17:25:18';
```

SQL1

UNION ALL

UNION→UNION ALL

```
SELECT t.SERIALNO
      ,t.USER_ID
      ,t.CHNL_CODE
      ,t.CHNL_NM
      ,t.USER_TYPE_CD AS USER_CATAGORY
      ,t.CNTMNG_SWFTNO
      ,t.BLST_TIME
FROM t_mm_blacklist t
WHERE t.USER_TYPE_CD = '0'
      AND t.USER_ID = '15485599967'
UNION ALL
SELECT t.SERIALNO
      ,t.USER_ID
      ,t.CHNL_CODE
      ,t.CHNL_NM
      ,t.USER_TYPE_CD AS USER_CATAGORY
      ,t.CNTMNG_SWFTNO
      ,t.BLST_TIME
FROM t_mm_blacklist t
WHERE t.USER_ID = '15485599967'
      AND t.USER_TYPE_CD = '1'
      AND t.BLST_TIME > '2017-05-16 17:25:18';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	t	NULL	ref	IDX_MM_BLACKLIST_USERID	IDX_MM_BLACKLIST_USERID	602	const	1	30.95	Using where
2	UNION	t	NULL	ref	IDX_MM_BLACKLIST_USERID	IDX_MM_BLACKLIST_USERID	602	const	1	36.73	Using where
NULL	UNION RESULT	<union1,2>	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	Using temporary

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	t	NULL	ref	IDX_MM_BLACKLIST_USERID	IDX_MM_BLACKLIST_USERID	602	const	1	30.95	Using where
2	UNION	t	NULL	ref	IDX_MM_BLACKLIST_USERID	IDX_MM_BLACKLIST_USERID	602	const	1	36.73	Using where

案例分享——SQL改写N

```
SELECT /*94*/ a.PARA_NO
,a.PARA_NM
,a.FST_CHAR_ATTR_VAL
,a.SECD_CHAR_ATTR_VAL
,a.EFF_TIME
,a.INVLD_TIME
,b.CHNL_NM
,b.SECD_CHAR_ATTR_VAL AS insertName
,a.MODF_USER_ID
,a.MODF_TIME
FROM t_mm_commpara_bs a
,t_mm_chnl b
,t_mm_chnl_insertcode c
WHERE PARA_SEQNO = '4'
AND a.FST_CHAR_ATTR_VAL = b.CHNL_NM
AND a.SECD_CHAR_ATTR_VAL = b.FST_CHAR_ATTR_VAL
AND a.SECD_CHAR_ATTR_VAL = c.CHNL_NM
AND c.PROV_CODE IN (
'100'
,'000'
)
AND a.FST_CHAR_ATTR_VAL = '000010101'
AND a.SECD_CHAR_ATTR_VAL = '1009001'
AND a.PARA_NO = '0000068'
AND (
a.EFF_TIME > '2016-07-21 08:00:00'
OR a.EFF_TIME = '2017-02-27 09:38:25'
)
AND a.invld time = '2017-02-27 09:39:00'
ORDER BY a.PARA_NO
,a.FST_CHAR_ATTR_VAL
,a.SECD CHAR ATTR VAL LIMIT 0,4;
```

```
AND t1.STAFFID IS NULL
AND t1.CTIID = #{ctiId}
AND t1.VDNID = #{vdnId}
AND t1.WORKNO = #{workNo}
AND t1.STAFFID = #{staffId}
```

无效的ORDER BY

效率比'='差太多

相差十万八千里

```
WHERE t1.TPL_CNTT = t2.TMPLT_ID
AND t1.TENANT_ID = ?
AND t1.PROV_CODE = ? LIMIT 0, 10

mysql> desc t_sr_cfg_build_order_num_forewarn;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| TPL CNTT | varchar(4000) | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+

mysql> desc t_sr_cfg_tpl;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| TMPLT_ID | varchar(40) | NO | PRI | NULL | |
+-----+-----+-----+-----+-----+-----+
```

总结

