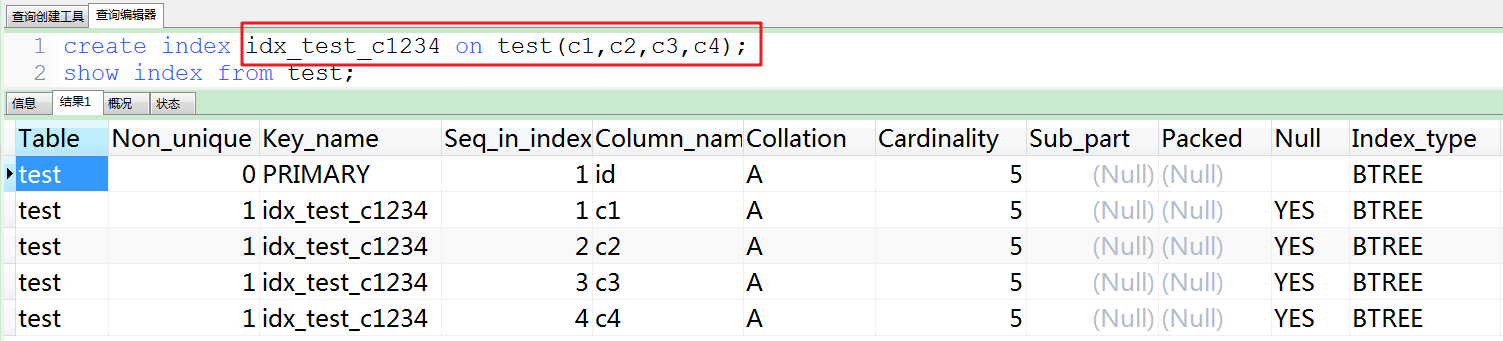
**创建test表（测试表）**

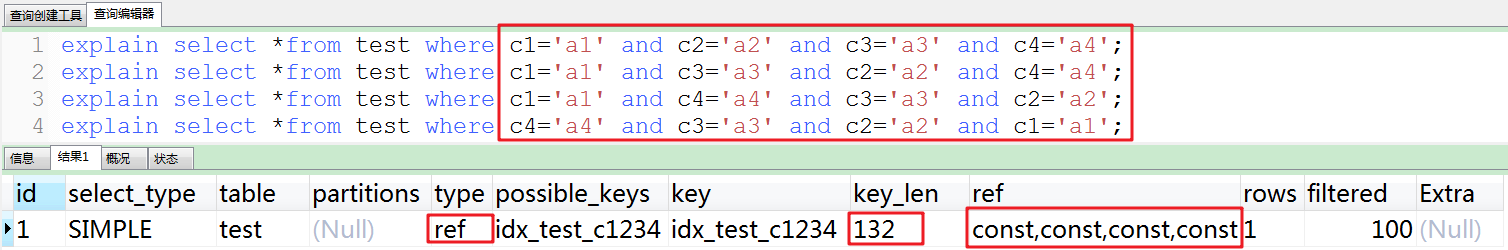
drop table if exists test;
create table test(
id int primary key auto\_increment,
c1 varchar(**10**),
c2 varchar(**10**),
c3 varchar(**10**),
c4 varchar(**10**),
c5 varchar(**10**)
) ENGINE=INNODB default CHARSET=utf8;
insert into test(c1,c2,c3,c4,c5) values('a1','a2','a3','a4','a5');
insert into test(c1,c2,c3,c4,c5) values('b1','b2','b3','b4','b5');
insert into test(c1,c2,c3,c4,c5) values('c1','c2','c3','c4','c5');
insert into test(c1,c2,c3,c4,c5) values('d1','d2','d3','d4','d5');
insert into test(c1,c2,c3,c4,c5) values('e1','e2','e3','e4','e5');

**创建索引**



**分析以下Case索引使用情况**

Case 1：



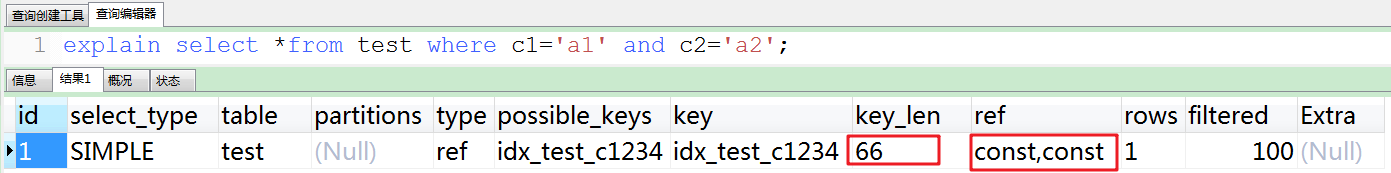
分析：

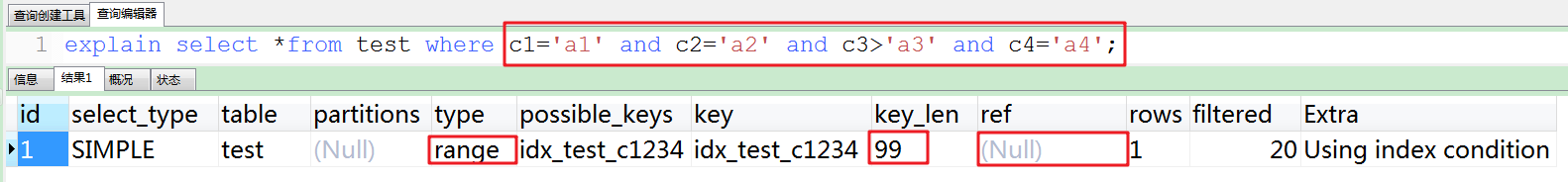
①创建复合索引的顺序为c1,c2,c3,c4。

②上述四组explain执行的结果都一样：type=ref，key\_len=132，ref=const,const,const,const。

结论：在执行常量等值查询时，改变索引列的顺序并不会更改explain的执行结果，因为mysql底层优化器会进行优化，但是推荐按照索引顺序列编写sql语句。

Case 2：



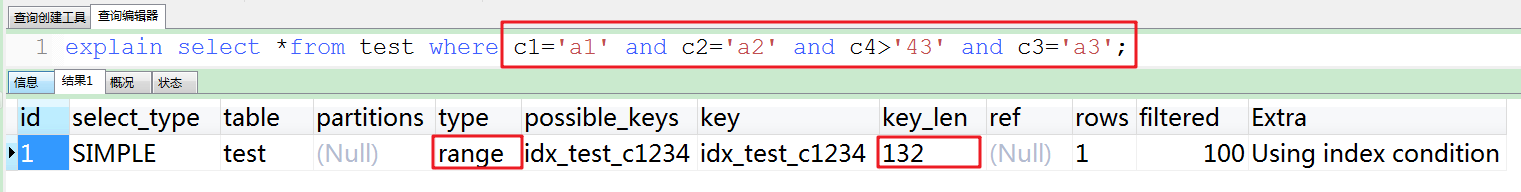


分析：

当出现范围的时候，type=range，key\_len=99，比不用范围key\_len=66增加了，说明使用上了索引，但对比Case1中执行结果，说明c4上索引失效。

结论：范围右边索引列失效，但是范围当前位置（c3）的索引是有效的，从key\_len=99可证明。

Case 2.1：

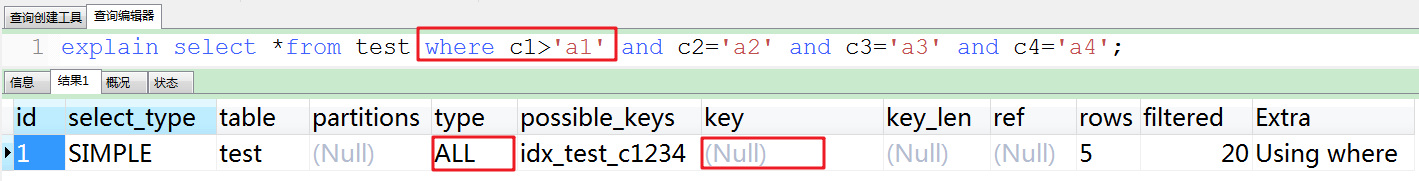


分析：

与上面explain执行结果对比，key\_len=132说明索引用到了4个，因为对此sql语句mysql底层优化器会进行优化：范围右边索引列失效（c4右边已经没有索引列了），注意索引的顺序（c1,c2,c3,c4），所以c4右边不会出现失效的索引列，因此4个索引全部用上。

结论：范围右边索引列失效，是有顺序的：c1,c2,c3,c4，如果c3有范围，则c4失效；如果c4有范围，则没有失效的索引列，从而会使用全部索引。

Case 2.2：



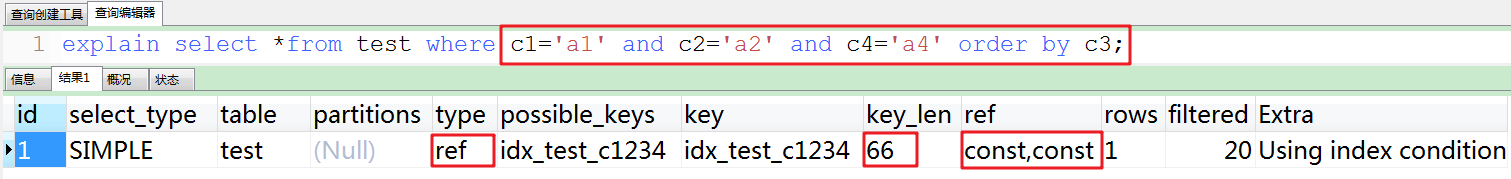
分析：

如果在c1处使用范围，则type=ALL，key=Null，索引失效，全表扫描，这里违背了最佳左前缀法则，带头大哥已死，因为c1主要用于范围，而不是查询。

解决方式使用覆盖索引。

结论：在最佳左前缀法则中，如果最左前列（带头大哥）的索引失效，则后面的索引都失效。

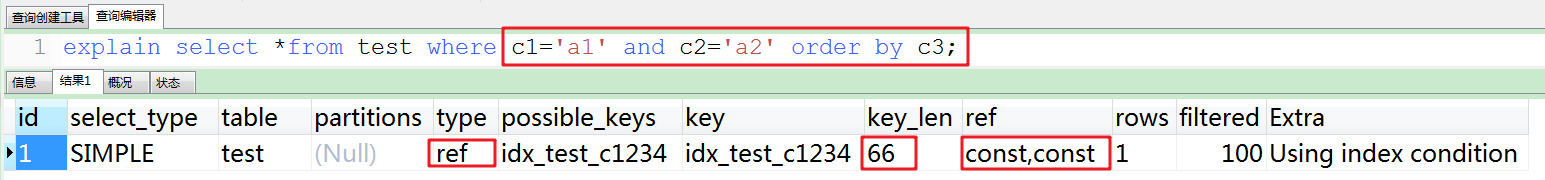
Case 3：



分析：

利用最佳左前缀法则：中间兄弟不能断，因此用到了c1和c2索引（查找），从key\_len=66，ref=const,const，c3索引列用在排序过程中。

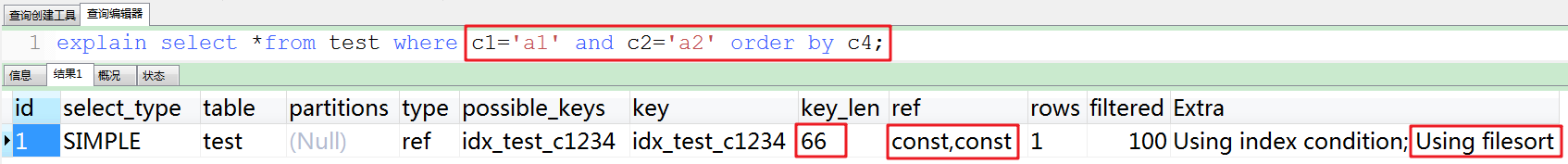
Case 3.1：



分析：

从explain的执行结果来看：key\_len=66，ref=const,const，从而查找只用到c1和c2索引，c3索引用于排序。

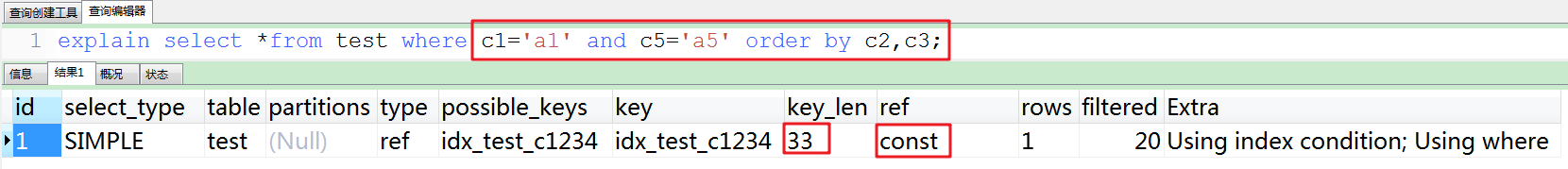
Case 3.2：



分析：

从explain的执行结果来看：key\_len=66，ref=const,const，查询使用了c1和c2索引，由于用了c4进行排序，跳过了c3，出现了Using filesort。

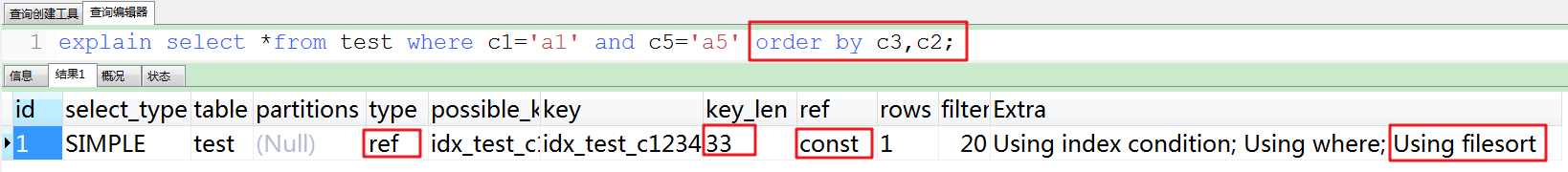
Case 4：



分析：

查找只用到索引c1，c2和c3用于排序，无Using filesort。

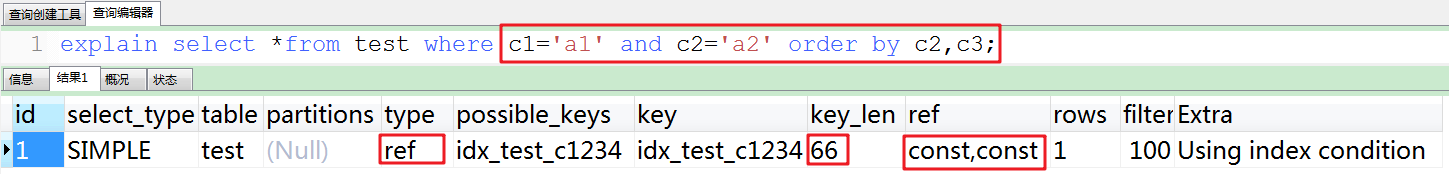
Case 4.1：

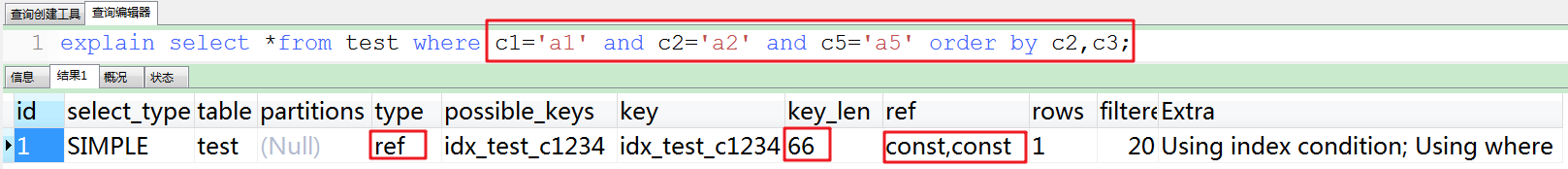


分析：

和Case 4中explain的执行结果一样，但是出现了Using filesort，因为索引的创建顺序为c1,c2,c3,c4，但是排序的时候c2和c3颠倒位置了。

Case 4.2：

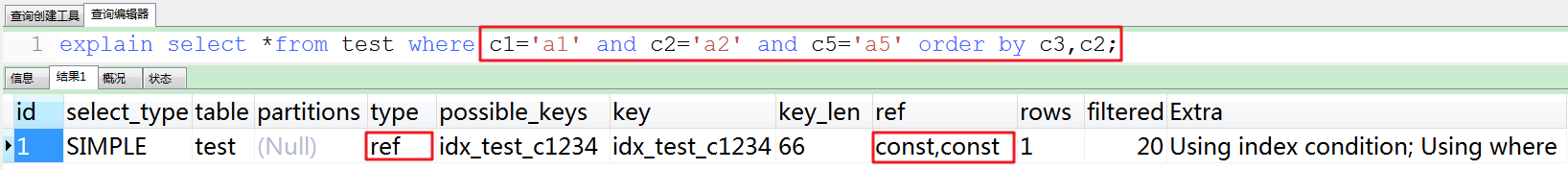




分析：

在查询时增加了c5，但是explain的执行结果一样，因为c5并未创建索引。

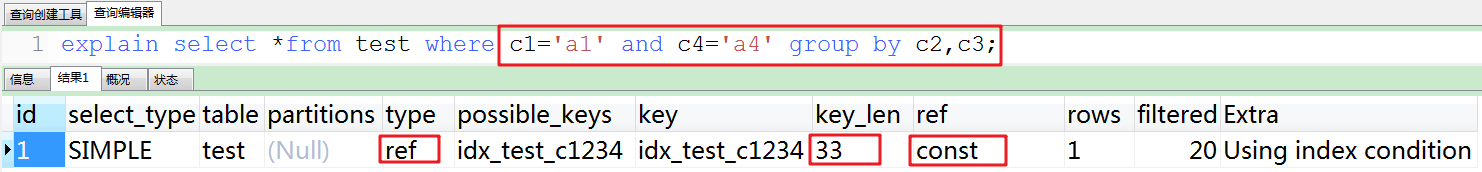
Case 4.3：



分析：

与Case 4.1对比，在Extra中并未出现Using filesort，因为c2为常量，在排序中被优化，所以索引未颠倒，不会出现Using filesort。

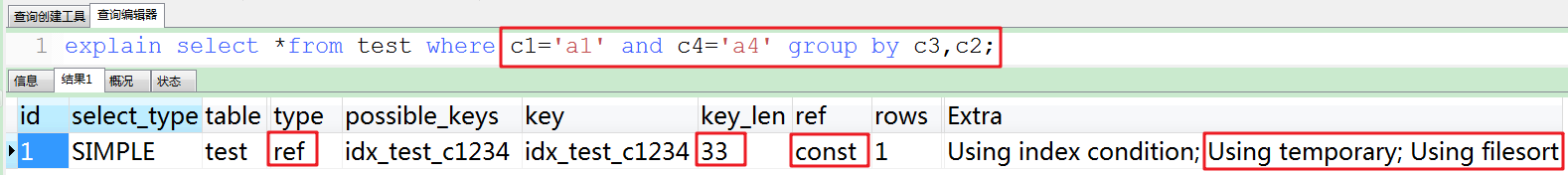
Case 5：



分析：

只用到c1上的索引，因为c4中间间断了，根据最佳左前缀法则，所以key\_len=33，ref=const，表示只用到一个索引。

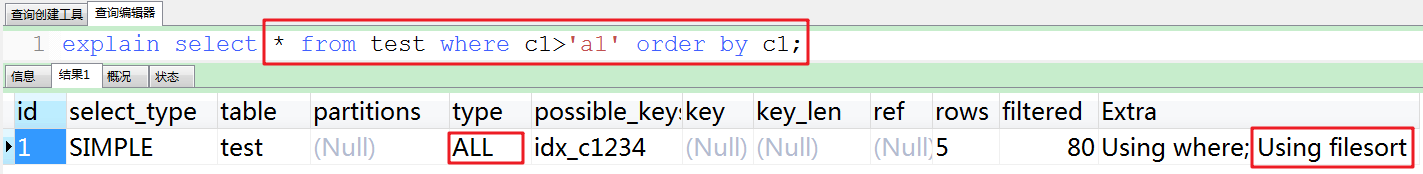
Case 5.1：



分析：

对比Case 5，在group by时交换了c2和c3的位置，结果出现Using temporary和Using filesort，极度恶劣。原因：c3和c2与索引创建顺序相反。

Case 6：

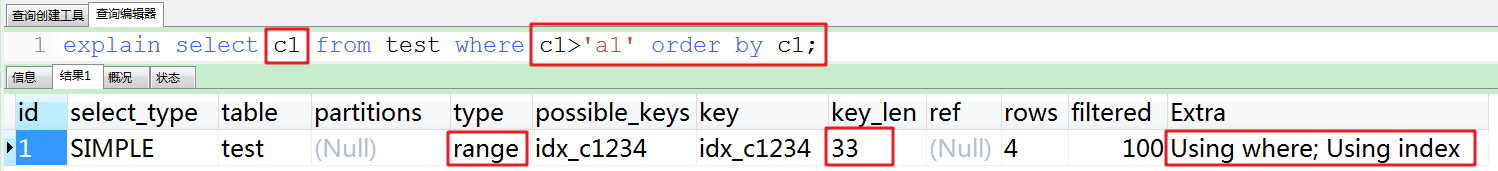


分析：

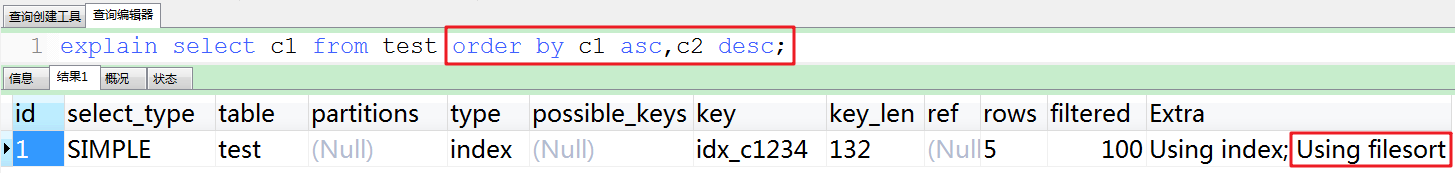
①在c1,c2,c3,c4上创建了索引，直接在c1上使用范围，导致了索引失效，全表扫描：type=ALL，ref=Null。因为此时c1主要用于排序，并不是查询。

②使用c1进行排序，出现了Using filesort。

③解决方法：使用覆盖索引。



Case 7：



分析：

虽然排序的字段列与索引顺序一样，且order by默认升序，这里c2 desc变成了降序，导致与索引的排序方式不同，从而产生Using filesort。

Case 8：

EXPLAIN extended select c1 from test where c1 in ('a1','b1') ORDER BY c2,c3;

clipboard.png

分析：

对于排序来说，多个相等条件也是范围查询

**总结：**

①MySQL支持两种方式的排序filesort和index，Using index是指MySQL扫描索引本身完成排序。index效率高，filesort效率低。

②order by满足两种情况会使用Using index。

#1.order by语句使用索引最左前列。

#2.使用where子句与order by子句条件列组合满足索引最左前列。

③尽量在索引列上完成排序，遵循索引建立（索引创建的顺序）时的最佳左前缀法则。

④如果order by的条件不在索引列上，就会产生Using filesort。

⑤group by与order by很类似，其实质是先排序后分组，遵照索引创建顺序的最佳左前缀法则。注意where高于having，能写在where中的限定条件就不要去having限定了。

**通俗理解口诀：**

   全值匹配我最爱，最左前缀要遵守；

   带头大哥不能死，中间兄弟不能断；

   索引列上少计算，范围之后全失效；

   LIKE百分写最右，覆盖索引不写星；

   不等空值还有or，索引失效要少用。

**补充：in和exsits优化**

原则：**小表驱动大表**，即小的数据集驱动大的数据集

in：当B表的数据集必须小于A表的数据集时，in优于exists

select \* from A where id in (select id from B)
#等价于：
　　for select id from B
　　for select \* from A where A.id = B.id

exists：当A表的数据集小于B表的数据集时，exists优于in

　　将主查询A的数据，放到子查询B中做条件验证，根据验证结果（true或false）来决定主查询的数据是否保留

select \* from A where exists (select **1** from B where B.id = A.id)
#等价于
for select \* from A
for select \* from B where B.id = A.id
#A表与B表的ID字段应建立索引

1、EXISTS (subquery)只返回TRUE或FALSE,因此子查询中的SELECT \* 也可以是SELECT 1或select X,官方说法是实际执行时会忽略SELECT清单,因此没有区别

2、EXISTS子查询的实际执行过程可能经过了优化而不是我们理解上的逐条对比

3、EXISTS子查询往往也可以用JOIN来代替，何种最优需要具体问题具体分析