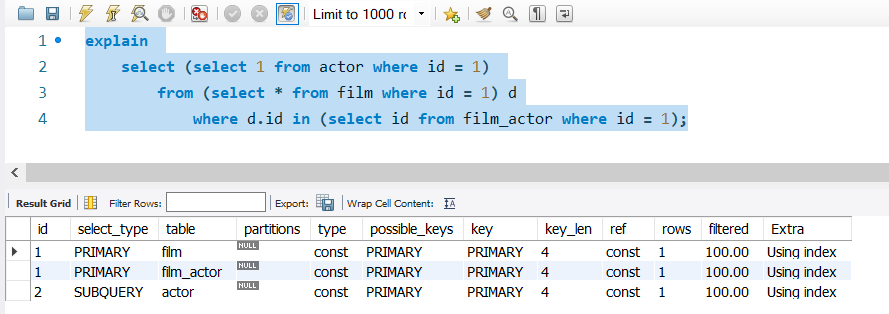
## Explain说明

通过sql查看执行计划



其中包含以下字段

Id、select\_type、table、partitions、type、possible\_keys、key、key \_len、ref、rows、

filtered、Extra。以下逐一说明。

### Id

复杂Sql中各sql执行顺序：

1. 执行顺序按由大到小执行,
2. 如果id一样则由上往下执行
3. Null的最后才执行

### select\_type

就是查询类型

**Simple**:简单查询，不包含子查询等复杂情况，例如select \* from table t where t.a=’a’;

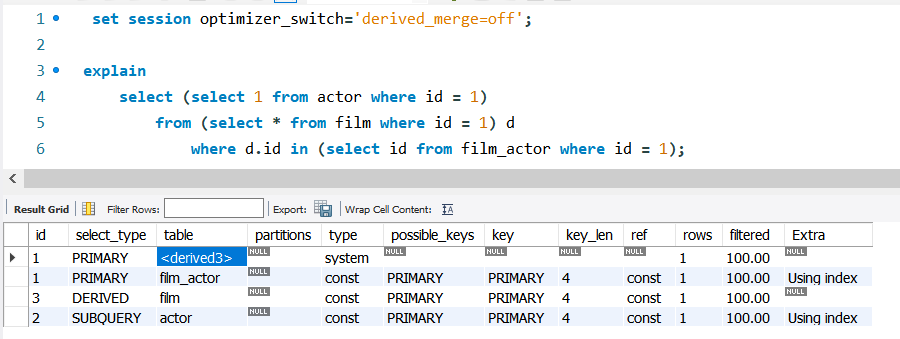
**primary** 主要查询，简单说就是执行最后拼装完成的sql

**Subquery**包含在select语句中的子查询

**Derived** 包含from后面的子查询，mysql会把查询结果放在一张临时表(也叫派生表Derived)

以下是例子，首先关闭mysql5.7新特性对衍生表的合并优化

set session optimizer\_switch='derived\_merge=off';



PS:记得还原设置：

set session optimizer\_switch='derived\_merge=on';

### Table

表示当前explain查询的表；

**DerivedN** 就是临时表，上面select\_type的Derived3表示本临时表依赖id为3的语句，需要先执行id=3的语句

当有 union 时，UNION RESULT 的 table 列的值为<union1,2>，1和2表示参与 union 的 select 行id。

### Partitions

### Type

关于本次查询的关联类型或者访问类型，即mysql决定如何查找表中行，查找数据行的大概范围，从优到差排序是：

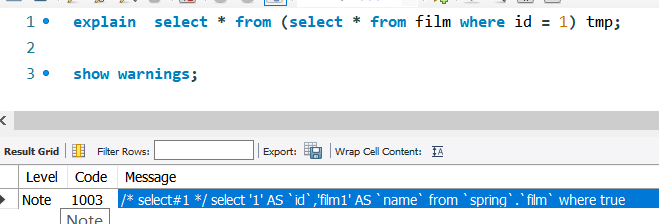
**System>conts>eq\_ref>ref>range>index>ALL**

一般最低要求是range，最好是ref，否则就要优化以下逐个分析：

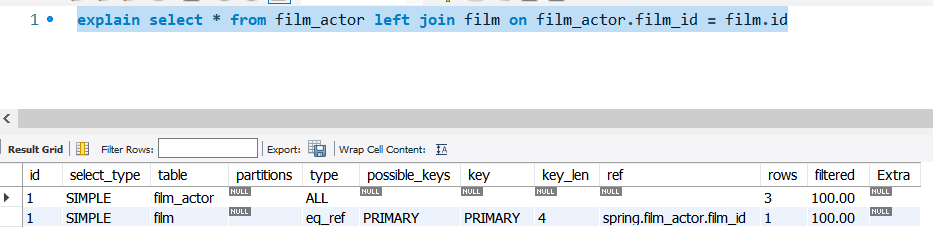
**NULL**表示mysql再优化阶段已经知道结果，而并没有执行查询表或者索引。典型就是找索引列的最大/小值。

Selet max(a) from table；

**const, system**：mysql能对查询的某部分进行优化并将其转化成一个常量（可以看show warnings 的结果）。用于primary key 或 unique key 的所有列与常数比较时，所以表最多有一个匹配行，读取1次，速度比较快。system是const的特例，表里只有一条元组匹配时为system。

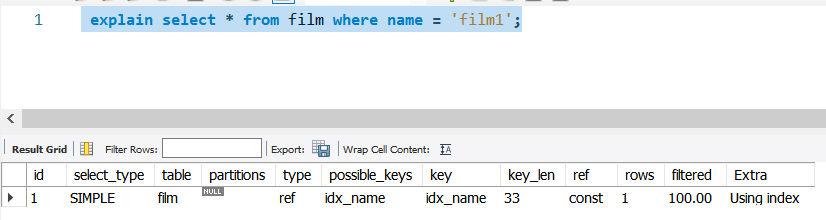


**eq\_ref**：primary key 或 unique key 索引的所有部分被连接使用 ，最多只会返回一条符合条件的记录。这可能是在const 之外最好的联接类型了，简单的 select 查询不会出现**eq\_ref**。

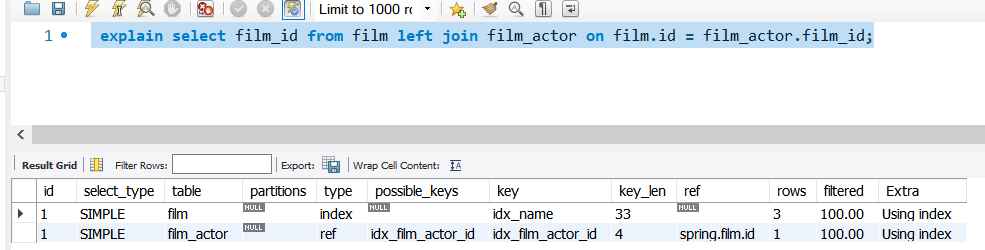


**ref**：相比 eq\_ref，不使用唯一索引，而是使用普通索引或者唯一性索引的部分前缀，索引要和某个值相比较，可能会找到多个符合条件的行。

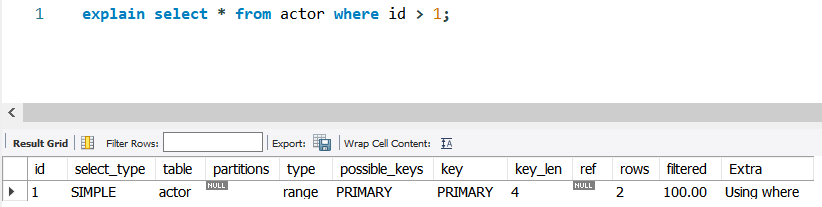
1，简单 select 查询，name是普通索引（非唯一索引）



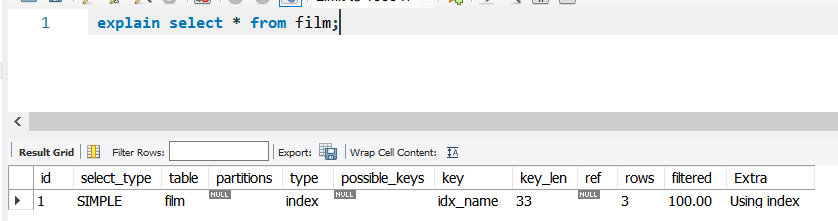
1. 关联表查询，idx\_film\_actor\_id是film\_id和actor\_id的联合索引，这里使用到了film\_actor的左边前缀film\_id部分。



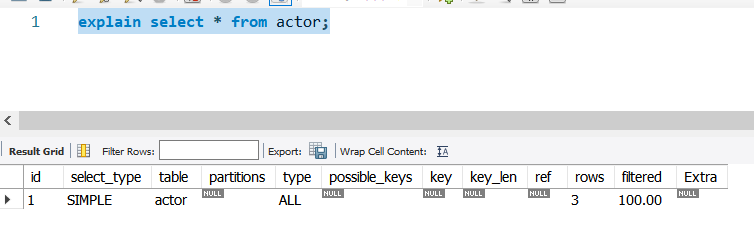
**range**：范围扫描通常出现在 in(), between ,> ,<, >= 等操作中。使用一个索引来检索给定范围的行。



**index**：扫描全索引就能拿到结果，一般是扫描某个二级索引，这种扫描不会从索引树根节点开始快速查找，而是直接对二级索引的叶子节点遍历和扫描，速度还是比较慢的，这种查询一般为使用覆盖索引，二级索引一般比较小，所以这种通常比ALL快一些。



ALL：即全表扫描，扫描你的聚簇索引的所有叶子节点。通常情况下这需要增加索引来进行优化了。



### possible\_keys

本次查询可能使用索引；

有时候有值，但是key为null则表示表数据少，没必要通过索引查询；

如果possible\_keys为null则可以考虑针对查询条件新增索引，并通过explain查看效率；

### Key

这一列显示mysql实际采用哪个索引来优化对该表的访问。

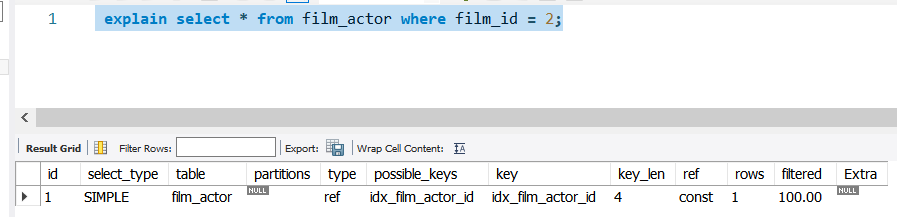
如果没有使用索引，则该列是 NULL。如果想强制mysql使用或忽视possible\_keys列中的索引，在查询中使用 force index、ignore index。

### key\_len

这一列显示了mysql在索引里使用的字节数，通过这个值可以算出具体使用了索引中的哪些列。

举例来说，film\_actor的联合索引 idx\_film\_actor\_id 由 film\_id 和 actor\_id 两个int列组成，并且每个int是4字节。通

过结果中的key\_len=4可推断出查询使用了第一个列：film\_id列来执行索引查找。



key\_len计算规则如下：

字符串，char(n)和varchar(n)，5.0.3以后版本中，n均代表字符数，而不是字节数，如果是utf-8，一个数字或字母占1个字节，一个汉字占3个字节。

* char(n)：如果存汉字长度就是 3n 字节
* varchar(n)：如果存汉字则长度是 3n + 2 字节，加的2字节用来存储字符串长度，因为varchar是变长字符串

数值类型

* tinyint：1字节
* smallint：2字节
* int：4字节
* bigint：8字节

时间类型

* date：3字节
* timestamp：4字节
* datetime：8字节

如果字段允许为 NULL，需要1字节记录是否为 NULL

索引最大长度是768字节，当字符串过长时，mysql会做一个类似左前缀索引的处理，将前半部分的字符提取出来做索引。

### Ref

这一列显示了在key列记录的索引中，表查找值所用到的列或常量，常见的有：const（常量），字段名（例：film.id）

### Rows

这一列是mysql估计要读取并检测的行数，注意这个不是结果集里的行数。

### Filtered

这里的filtered表示通过查询条件获取的最终记录行数占通过type字段指明的搜索方式搜索出来的记录行数的百分比。

filtered值越大表示索引越有效。

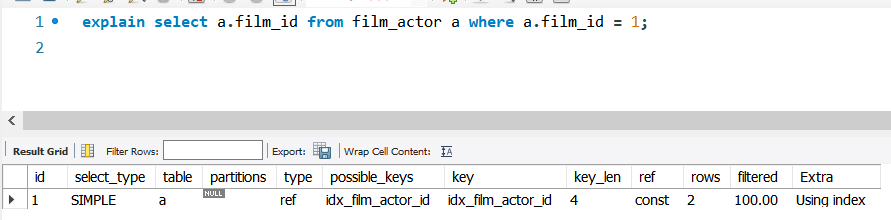
### Extra

这一列展示的是额外信息。常见的重要值如下：

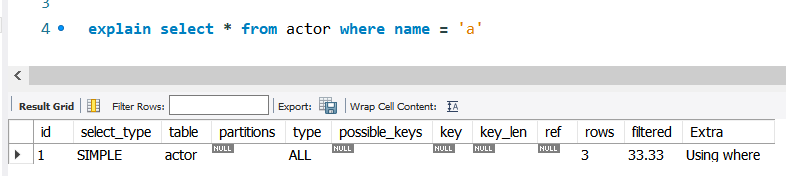
**Using index：使用覆盖索引**

覆盖索引定义：mysql执行计划explain结果里的key有使用索引，如果select后面查询的字段都可以从这个索引的树中获取，这种情况一般可以说是用到了覆盖索引，extra里一般都有using index；

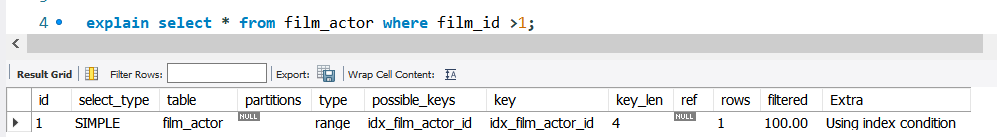
覆盖索引一般针对的是辅助索引，整个查询结果只通过辅助索引就能拿到结果，不需要通过辅助索引树找到主键，再通过主键去主键索引树里获取其它字段值。



**Using where：使用 where 语句来处理结果，**并且查询的列未被索引覆盖

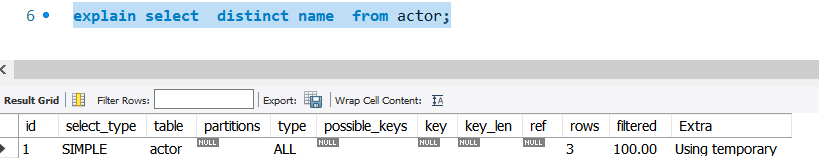


**Using index condition**：查询的列不完全被索引覆盖，where条件中是一个前导列的范围



**Using temporary**：mysql需要创建一张临时表来处理查询。出现这种情况一般是要进行优化的，首先是想到用索引来优化。

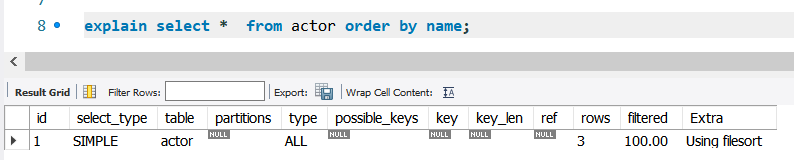
例如 actor.name没有索引，此时创建了张临时表来distinct



解决方案:name添加索引；

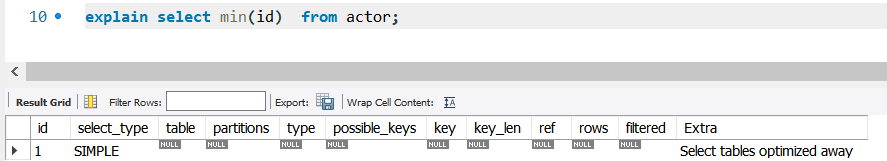
**Using filesort**：将用外部排序而不是索引排序，数据较小时从内存排序，否则需要在磁盘完成排序。这种情况下一般也是要考虑使用索引来优化的。

actor.name未创建索引，会浏览actor整个表，保存排序关键字name和对应的id，然后排序name并检索行记录。



解决方案:name添加索引；

**Select tables optimized away**：使用某些聚合函数（比如 max、min）来访问存在索引的某个字段。



## 优化实践

创建员工表employees。

### 创表语句

DROP TABLE IF EXISTS employees;

CREATE TABLE `employees` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(24) NOT NULL DEFAULT '' COMMENT '姓名',

`age` int(11) NOT NULL DEFAULT '0' COMMENT '年龄',

`position` varchar(20) NOT NULL DEFAULT '' COMMENT '职位',

`hire\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP COMMENT '入职时间',

PRIMARY KEY (`id`),

KEY `idx\_name\_age\_position` (`name`,`age`,`position`) USING BTREE

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8 COMMENT='员工记录表';

INSERT INTO employees(name,age,position,hire\_time) VALUES('LiLei',22,'manager',NOW());

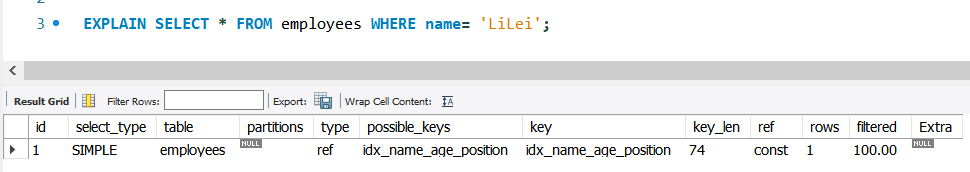
INSERT INTO employees(name,age,position,hire\_time) VALUES('HanMeimei',23,'dev',NOW());

INSERT INTO employees(name,age,position,hire\_time) VALUES('Lucy',23,'dev',NOW());

select \* from employees;

### 全值匹配

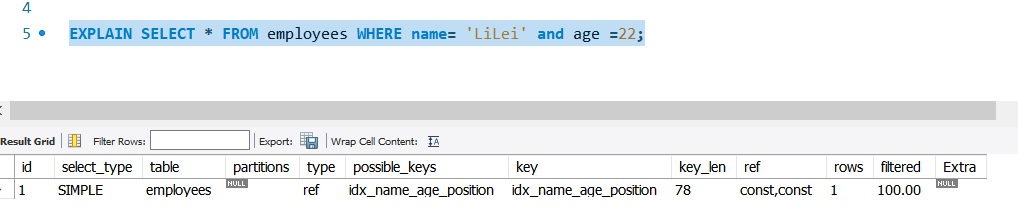
情况一：



Name是联合索引的第一个字段，这里key看到使用了该索引，然后key\_len是74的算法是：

`name` varchar(24)，24\*3+2=74

情况二：



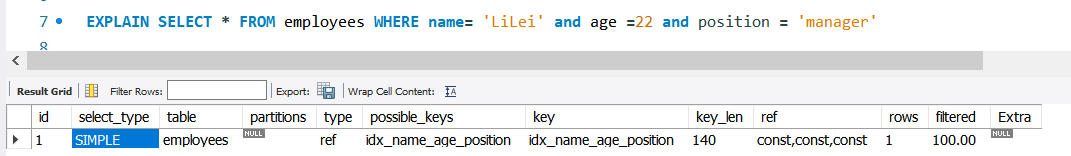
Name、age是联合索引的第一、二个字段，这里key看到使用了该索引，然后key\_len是78的算法是：

`name` varchar(24)，24\*3+2=74

`age` int(11) 是4

所有 74+4=78

情况三：



Name、age、position是联合索引的第一、二、三个字段，这里key看到使用了该索引，然后key\_len是140的算法是：

`name` varchar(24) 24\*3+2=74

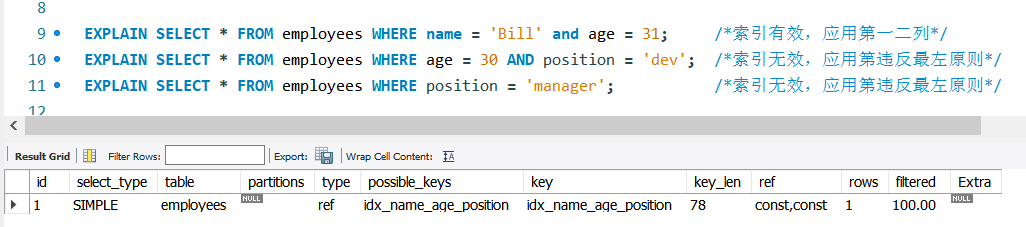
`age` int(11) 是4

`position` varchar(20) 20\*3+2=62

所有 74+4+62=140

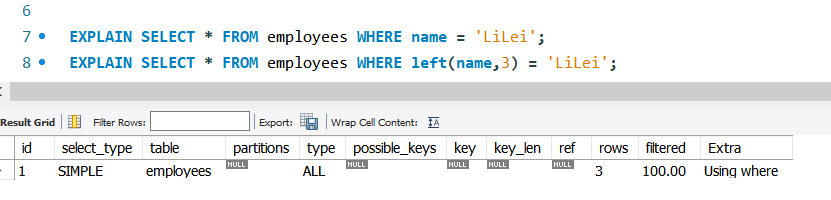
### 最左前缀法则

如果索引了多列，要遵守最左前缀法则。指的是查询从索引的最左前列开始并且不跳过索引中的列。



### 不在索引字段做任何操作

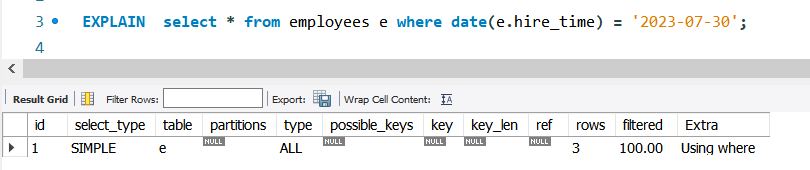
包含（计算、函数、（自动or手动）类型转换。



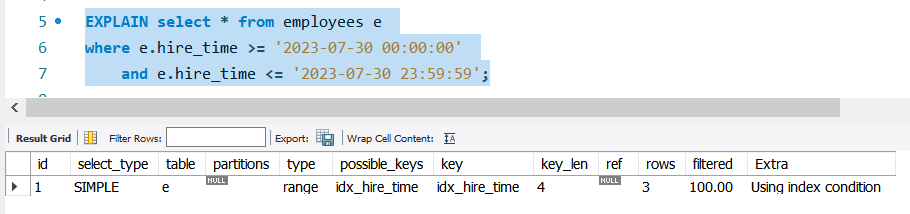
给hire\_time增加一个普通索引：

ALTER TABLE `employees` ADD INDEX `idx\_hire\_time` (`hire\_time`) USING BTREE ;

经过函数转换的等于查询，索引无效；



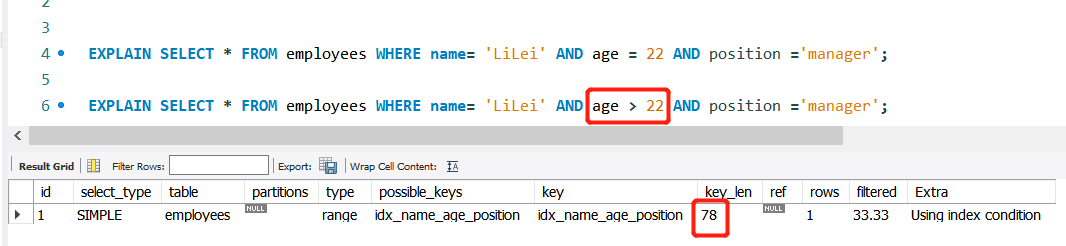
但是通过范围查询，索引则可能有效



实验完毕，删除索引

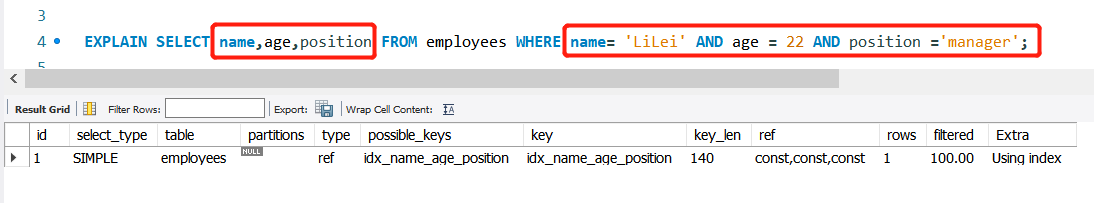
ALTER TABLE `employees` DROP INDEX `idx\_hire\_time`;

### 存储引擎不能使用索引中范围条件右边的列



范围查询后，就不符合索引的排列顺序，所以导致第三个索引无效；

### 尽量使用覆盖索引

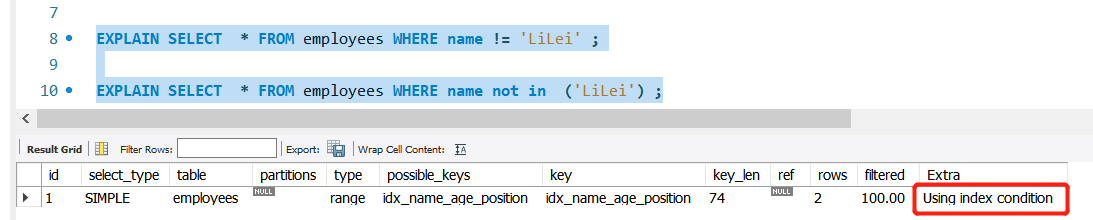


只访问索引的查询（索引列包含查询列）），减少 select \* 语句

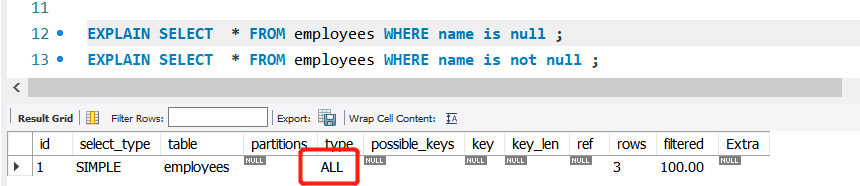
### 不使用否定条件

不等于（！=或者<>），not in ，not exists 的时候无法使用索引会导致全表扫描。

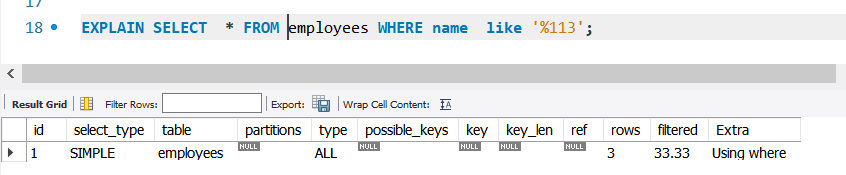
< 小于、 > 大于、 <=、>= 这些，mysql内部优化器会根据检索比例、表大小等多个因素整体评估是否使用索引。



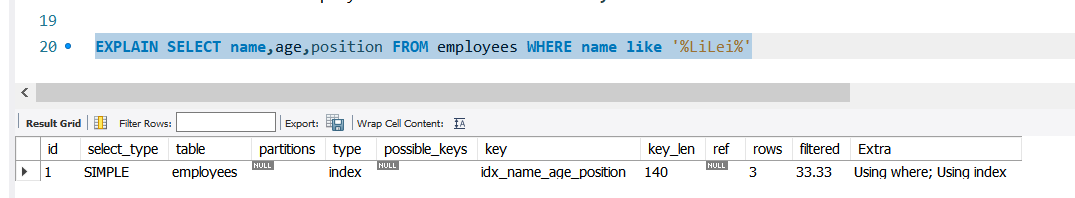
### is null,is not null 一般情况下也无法使用索引



### like开头导致无法使用索引

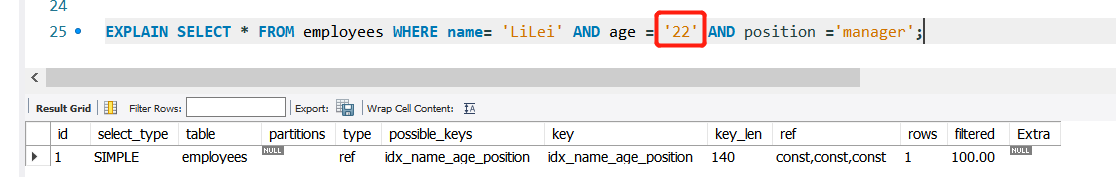


如果使用过覆盖索引则是索引有效



如果不能使用覆盖索引则可能需要借助搜索引擎。

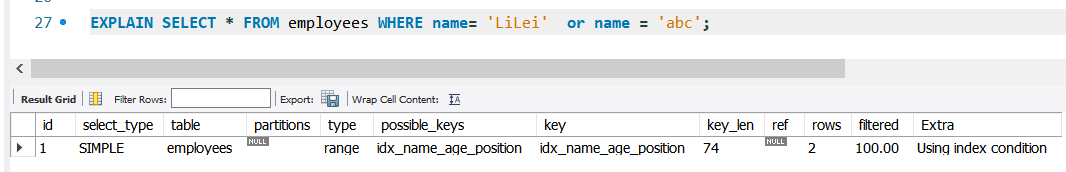
### 是否添加单引号也会影响索引



PS:看来8.0做了优化使索引有效

### 少用or或in

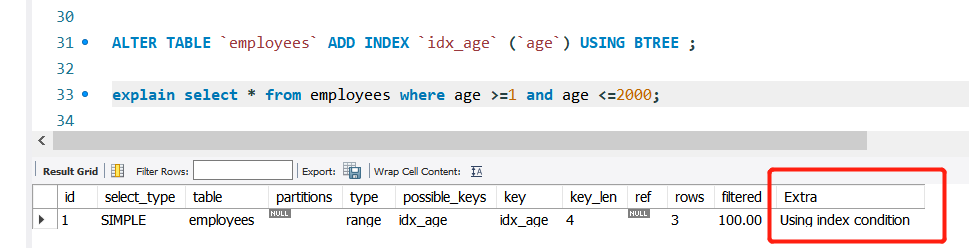
用它查询时，mysql不一定使用索引，mysql内部优化器会根据检索比例、表大小等多个因素整体评估是否使用索引，详见范围查询优化



PS:看来8.0做了优化使索引有效

### 范围查询优化

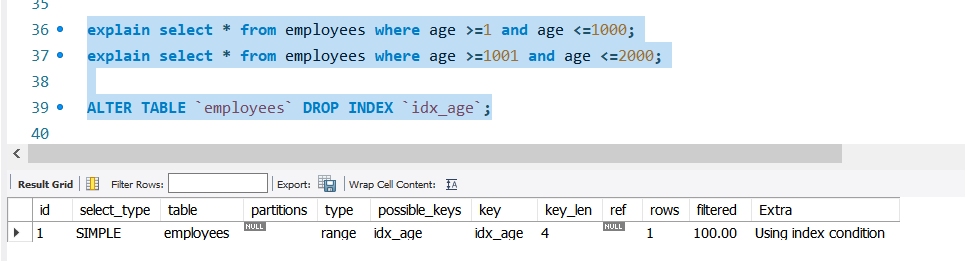
有时候搜索范围太多，索引会失效



但是本次还是用了，说明8.0做了优化。

另外没走索引原因：mysql内部优化器会根据检索比例、表大小等多个因素整体评估是否使用索引。比如这个例子，可能是由于单次数据量查询过大导致优化器最终选择不走索引

优化方法：可以将大的范围拆分成多个小范围：



### 总结

