```
 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';
```

在上面的示例中，我们创建了一个名为 `employees` 的表，包含了 `id`、`name`、`age`、`position` 和 `hire_time` 等字段，其中 `id` 字段作为主键，并创建了 `name`、`age` 和 `position` 组合的索引。

特别注意：**注意下建表时创建的组合索引，后文中不会再特意强调该索引。**

插入示例数据

接下来，小鱼向 `employees` 表中插入了一些示例数据，以便后续的查询和性能优化。

```
INSERT INTO employees(name,age,position,hire_time) VALUES('张三',22,'manager',NOW());
INSERT INTO employees(name,age,position,hire_time) VALUES('李四',23,'dev',NOW());
INSERT INTO employees(name,age,position,hire_time) VALUES('王五',23,'dev',NOW());
```

以上 SQL 语句向表中插入了三条员工记录，分别是张三、李四和王五，每条记录包括姓名、年龄、职位和入职时间等信息。

通过以上步骤，我们成功创建了示例数据表并插入了一些数据，接下来小鱼将通过实际查询来演示如何利用索引来提高数据库的查询性能。

SQL 实践索引优化

全值匹配

```
EXPLAIN SELECT * FROM employees WHERE name= '张三';
EXPLAIN SELECT * FROM employees WHERE name= '张三' AND age = 22;
EXPLAIN SELECT * FROM employees WHERE name= '张三' AND age = 22 AND position ='manager';
```

可以看到 `key_len` 和 `ref` 中看到使用了索引字段长度，以及表查找值所用到的常量。

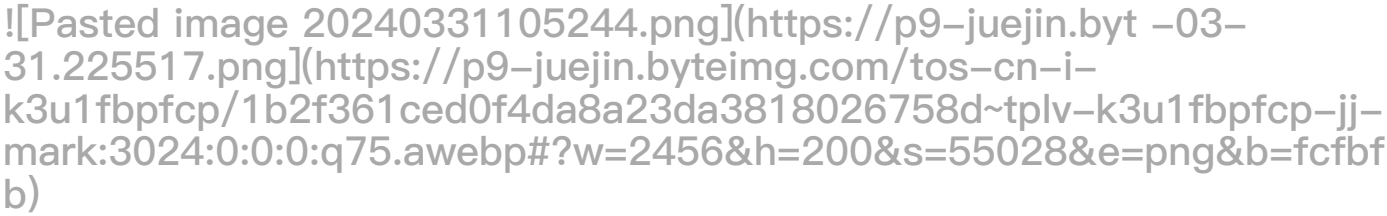
> `ref`：表示 key 列记录的索引中，表查找值所用到的字段或常量。常见的有：const（常量），字段名（例：film. Id）

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。

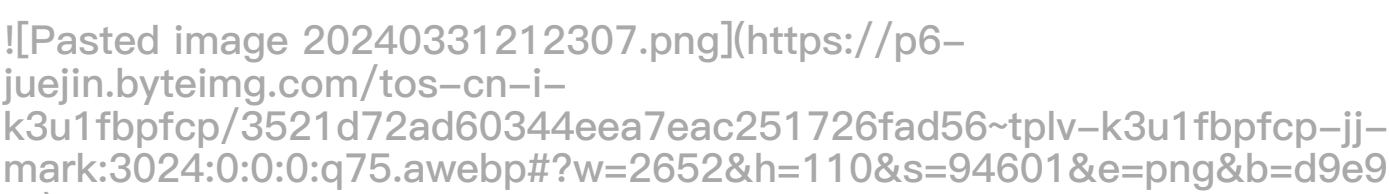
```
EXPLAIN SELECT * FROM employees WHERE name = '李四' AND age = 31;
EXPLAIN SELECT * FROM employees WHERE age = 23 AND position = 'dev';
EXPLAIN SELECT * FROM employees WHERE position = 'manager';
```

第一个语句我们可以看到使用了索引。



`is null` 或者 `is not null` 通常也不会走索引。

```
EXPLAIN SELECT * FROM employees WHERE name is null
```



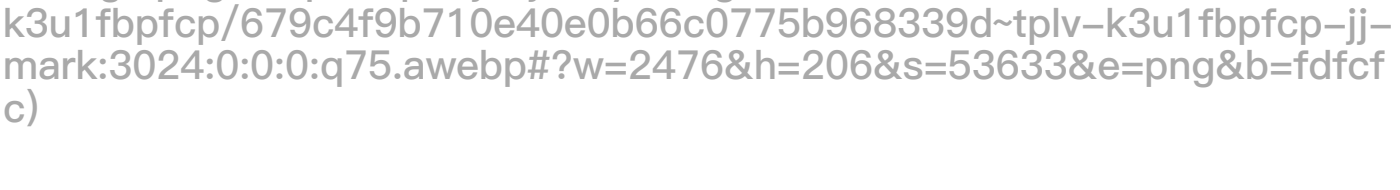
若是使用 >、<、<=、>=条件，mysql 内部会进行优化，评估是否使用索引。

语句使用 `in`、`or` 时不一定会使用索引

```
EXPLAIN SELECT * FROM employees WHERE name != '王五' or name = '张三';
```

同样，mysql 内部进行优化时，会根据索引比例、表大小等等因素，决定是否使用索引。

* **mysql 8 版本**也会走索引

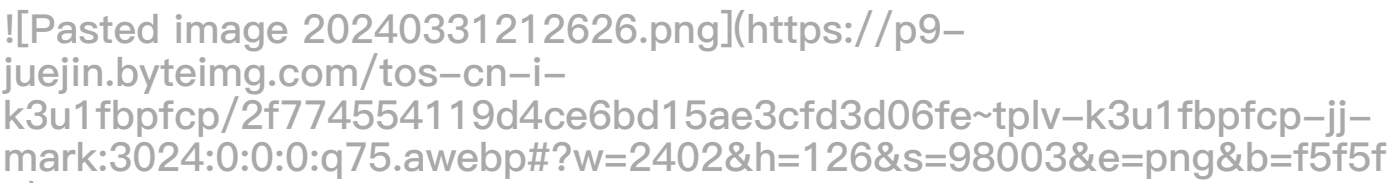


范围查找优化

我们先给 `age` 字段增加索引：

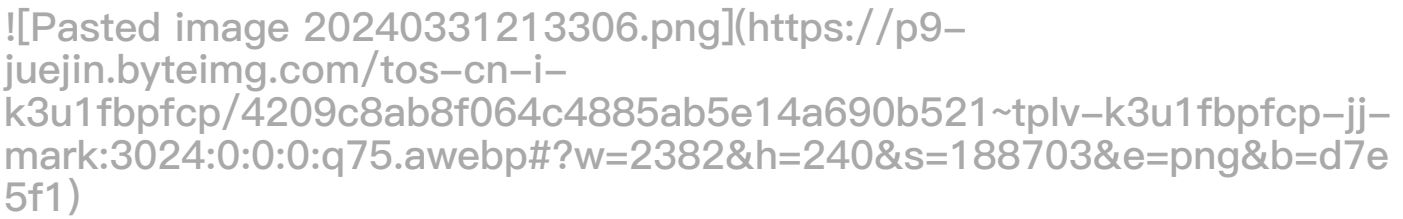
```
ALTER TABLE `employees` ADD INDEX `idx_age` (`age`) USING BTREE ;
```

```
EXPLAIN SELECT * FROM employees WHERE age >=1 AND age <= 50;
```



我们发现并没有走索引，原因同样也是 mysql 内部进行优化时，会根据索引比例、表大小等等因素，决定是否使用索引。

```
EXPLAIN SELECT * FROM employees WHERE age >= 1 AND age <= 22;
EXPLAIN SELECT * FROM employees WHERE age >= 23 AND age <= 50;
```



我们可以对查询范围进行拆分成多个小范围，这样也可以走索引。

删除索引

```
ALTER TABLE `employees` DROP INDEX `idx_age`;
```

总结
==