*select id,name where name='shenjian'*

*select id,name,sex*\* where name='shenjian'\*

**多查询了一个属性，为何检索过程完全不同？**

**什么是回表查询？**

**什么是索引覆盖？**

**如何实现索引覆盖？**

**哪些场景，可以利用索引覆盖来优化SQL？**

这些，这是今天要分享的内容。

*画外音：本文试验基于MySQL5.6-InnoDB。*

**一、什么是回表查询？**

这先要从InnoDB的索引实现说起，InnoDB有两大类索引：

* 聚集索引(clustered index)
* 普通索引(secondary index)

**InnoDB聚集索引和普通索引有什么差异？**

InnoDB**聚集索引**的叶子节点存储行记录，因此， InnoDB必须要有，且只有一个聚集索引：

（1）如果表定义了PK，则PK就是聚集索引；

（2）如果表没有定义PK，则第一个not NULL unique列是聚集索引；

（3）否则，InnoDB会创建一个隐藏的row-id作为聚集索引；

*画外音：所以PK查询非常快，直接定位行记录。*

InnoDB**普通索引**的叶子节点存储主键值。

*画外音：注意，不是存储行记录头指针，MyISAM的索引叶子节点存储记录指针。*

举个栗子，不妨设有表：

*t(id PK, name KEY, sex, flag);*

*画外音：id是聚集索引，name是普通索引。*

表中有四条记录：

*1, shenjian, m, A*

*3, zhangsan, m, A*

*5, lisi, m, A*

*9, wangwu, f, B*

image

两个B+树索引分别如上图：

（1）id为PK，聚集索引，叶子节点存储行记录；

（2）name为KEY，普通索引，叶子节点存储PK值，即id；

既然从普通索引无法直接定位行记录，那**普通索引的查询过程是怎么样的呢？**

通常情况下，需要扫码两遍索引树。

例如：

*select \* from t where name='lisi';*

**是如何执行的呢？**

image

如**粉红色**路径，需要扫码两遍索引树：

（1）先通过普通索引定位到主键值id=5；

（2）在通过聚集索引定位到行记录；

这就是所谓的**回表查询**，先定位主键值，再定位行记录，它的性能较扫一遍索引树更低。

**二、什么是索引覆盖\*\*\*\*(Covering index)\*\*\*\*？**

额，楼主并没有在MySQL的官网找到这个概念。

*画外音：治学严谨吧？*

借用一下SQL-Server官网的说法。

image

MySQL官网，类似的说法出现在explain查询计划优化章节，即explain的输出结果Extra字段为Using index时，能够触发索引覆盖。

image

不管是SQL-Server官网，还是MySQL官网，都表达了：只需要在一棵索引树上就能获取SQL所需的所有列数据，无需回表，速度更快。

**三、如何实现索引覆盖？**

常见的方法是：将被查询的字段，建立到联合索引里去。

仍是《[迅猛定位低效SQL？](https://links.jianshu.com/go?to=http%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMjM5ODYxMDA5OQ%3D%3D%26mid%3D2651962587%26idx%3D1%26sn%3Dd197aea0090ce93b156e0774c6dc3019%26chksm%3Dbd2d09078a5a801138922fb5f2b9bb7fdaace7e594d55f45ce4b3fc25cbb973bbc9b2deb2c31%26scene%3D21%23wechat_redirect" \t "_blank)》中的例子：

*create table user (*

*id int primary key,*

*name varchar(20),*

*sex varchar(5),*

*index(name)*

*)engine=innodb;*

第一个SQL语句：

image

*select id,name from user where name='shenjian';*

能够命中name索引，索引叶子节点存储了主键id，通过name的索引树即可获取id和name，无需回表，符合索引覆盖，效率较高。

*画外音，Extra：Using index。*

第二个SQL语句：

*image*

*select id,name,sex*\* from user where name='shenjian';\*

能够命中name索引，索引叶子节点存储了主键id，但sex字段必须回表查询才能获取到，不符合索引覆盖，需要再次通过id值扫码聚集索引获取sex字段，效率会降低。

*画外音，Extra：Using index condition。*

如果把(name)单列索引升级为联合索引(name, sex)就不同了。

*create table user (*

*id int primary key,*

*name varchar(20),*

*sex varchar(5),*

*index(name, sex)*

*)engine=innodb;*

image

可以看到：

*select id,name ... where name='shenjian';*

*select id,name,sex*\* ... where name='shenjian';\*

都能够命中索引覆盖，无需回表。

*画外音，Extra：Using index。*

**四、哪些场景可以利用索引覆盖来优化SQL？**

**场景1：全表count查询优化**

image

原表为：

*user(PK id, name, sex)；*

直接：

*select count(name) from user;*

不能利用索引覆盖。

添加索引：

*alter table user add key(name);*

就能够利用索引覆盖提效。

**场景2：列查询回表优化**

*select id,name,sex ... where name='shenjian';*

这个例子不再赘述，将单列索引(name)升级为联合索引(name, sex)，即可避免回表。

**场景3：分页查询**

*select id,name,sex ...* ***order by*** *name limit 500,100;*

将单列索引(name)升级为联合索引(name, sex)，也可以避免回表。

**InnoDB聚集索引普通索引**，**回表**，**索引覆盖**，希望这1分钟大家有收获。