一、不同职位对MySQL的技术要求

- 数据分析
  - 偏重于查询
  - 复杂查询的写法
- 程序员
  - 表的设计
  - 增删改查
  - 注重SQL性能
- DBA
  - MySQL服务器配置
  - SQL、服务性能、稳定性
  - 数据一致性

二、Python操作MySQL

- Python DB-API

2-1、常用模块

a、MySQLdb

- 也就是MySQL-python
- 底层C实现

b、mysql-connector

- MySQL官方提供。

c、pymysql

- Python实现
- Python V3+

2-2、基本操作

- 连接数据库

```
import pymysql
db = pymysql.connect(host='127.0.0.1', port=3306)
```

- 获取游标

```
cs = db.cursor()
```

- 选择数据库

```
db.select_db('test')
```

- 执行SQL

```
sql = 'show databases'
result = cs.execute(sql)
dbs = cs.fetchall()
```

- 关闭数据库连接

```
db.close()
```

2-3、ORM

- ORM：object relational mapping，对象映射关系
- 使用对象模型而不是sql来操作数据库
- sqlalchemy

2-4、练习

- 将抓取的数据导入到mysql中
- 编写一个DB类，封装好常用的MySQL操作

三、索引

- 对表中一列或多列的值进行排序
- 定义一种存储结构
- 快速检索到数据
- 存储引擎级实现

3-1、索引类型：

- 普通索引：MySQL中基本索引类型，没有什么限制，允许在定义索引的列中插入重复值和空值，纯粹为了查询数据更快一点。
- 唯一索引：索引列中的值必须是唯一的，但是允许为空值，
- 主键索引：是一种特殊的唯一索引，不允许有空值
- 组合索引：在表中的多个字段组合上创建的索引，只有在查询条件中使用了这些字段的左边字段时，索引才会被使用，使用组合索引时遵循最左前缀集合。
- 全文索引：通过大文本中的某个关键字，就能找到该字段所属的记录行
- 空间索引：空间索引是对空间数据类型的字段建立的索引，MySQL中的空间数据类型：GEOMETRY、POINT、LINESTRING、POLYGON。

相关概念：

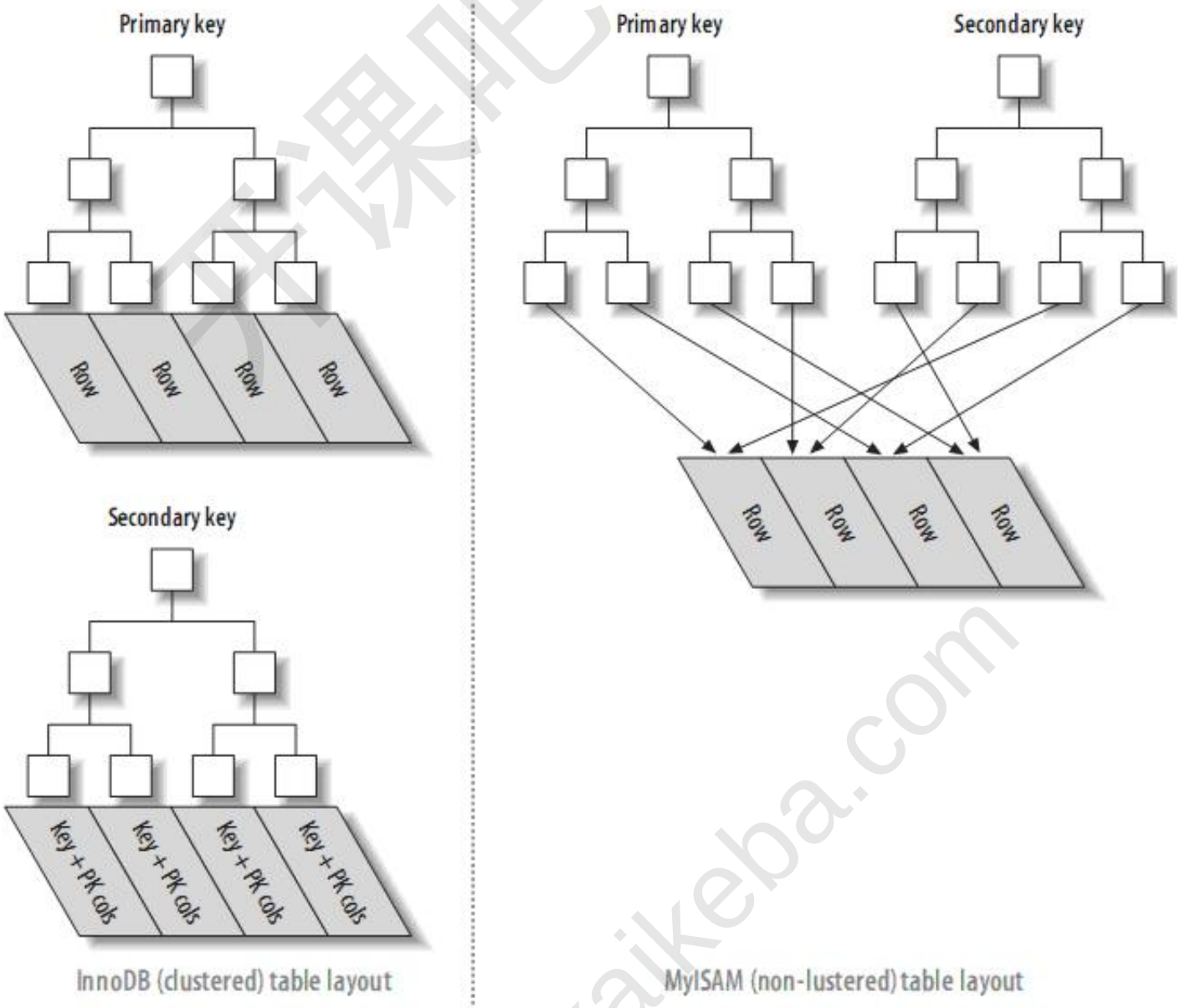
- 非聚簇索引：索引树的叶子节点存储数据的位置信息
- 聚簇索引：数据存储在索引树的叶子节点
- 覆盖索引：索引包含了（或覆盖了）查询语句中的字段与条件的数据

注意事项：

- 执行查询时，MySQL只能使用一个索引
- 提高了查询速度，降低了更新速度
- 不要使用like “%aaa%”操作，
- 越小的数据类型通常更好
- 简单的数据类型更好
- 尽量避免NULL
- 最左前缀：组合索引中的左边列

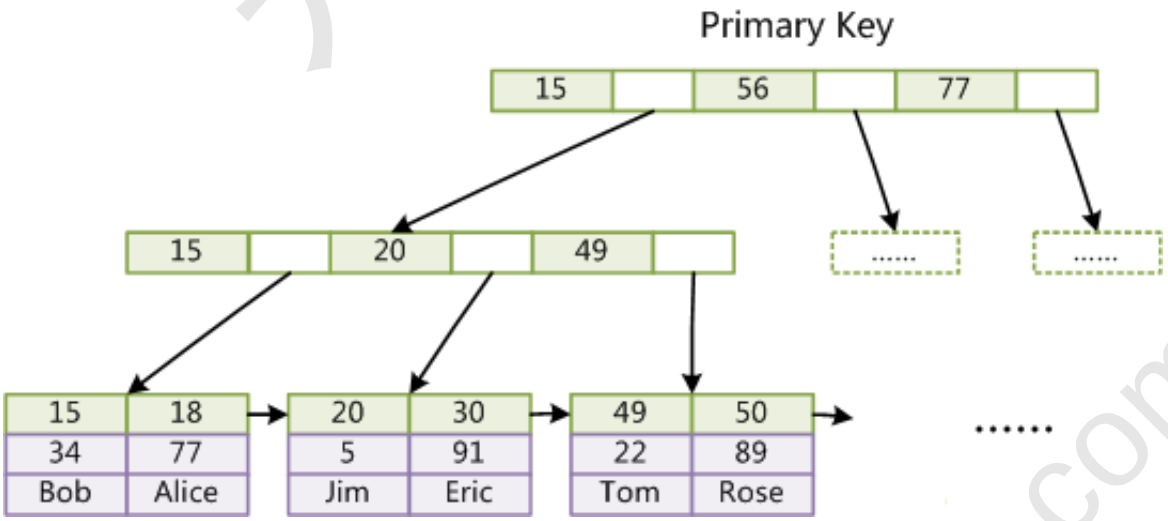
3-2、InnoDB及MyISAM索引结构

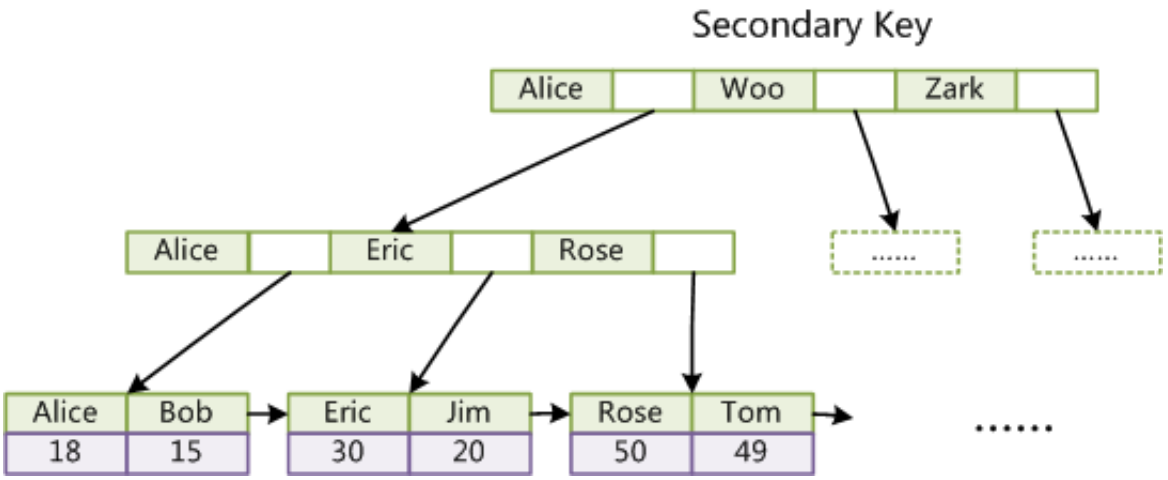
- MyISAM引擎：使用b+tree作为索引，叶子节点data存放的是记录地址。MyISAM中主索引与辅助索引形式是一样的，主索引要求key不能重复，辅助索引key可以重复。
- InnoDB引擎：与MyISAM索引与数据分开存放不同的是，InnoDB引擎数据文件本身就是一个索引，InnoDB引擎数据存放是按b+tree结构组织存放的，叶子节点包含全部的数据信息，InnoDB引擎辅助索引叶子节点存放的是主键。
  所以，InnoDB的普通索引，实际上会扫描两遍：
  第一遍，由普通索引找到PK；
  第二遍，由PK找到行记录；



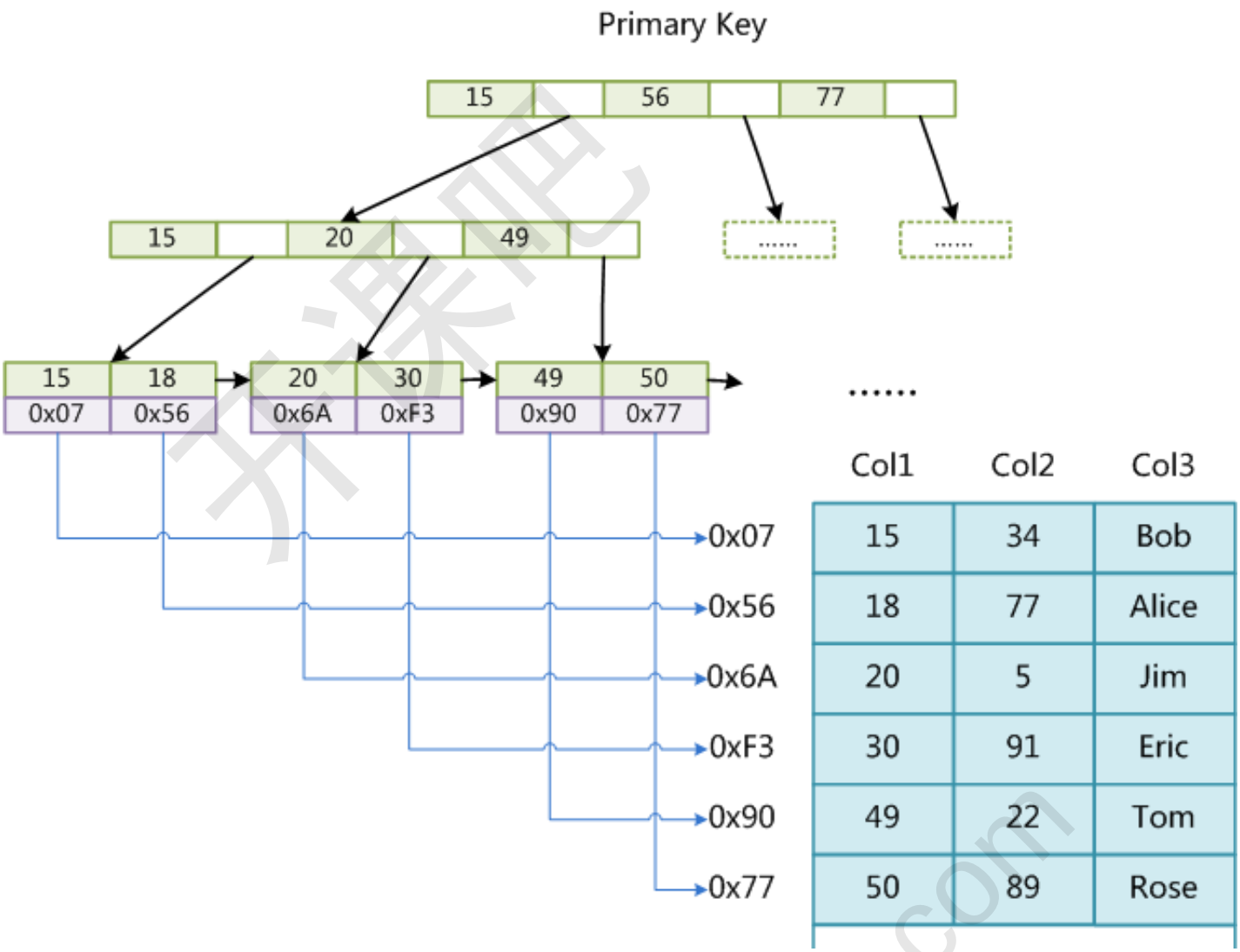
```
mysql> select * from tb;
+----+-----+-------+
| id | age | name  |
+----+-----+-------+
| 15 |  34 | Bob   |
| 18 |  77 | Alice |
| 20 |   5 | Jim   |
| 30 |  91 | Eric  |
| 49 |  22 | Tom   |
| 50 |  89 | Rose  |
+----+-----+-------+
```

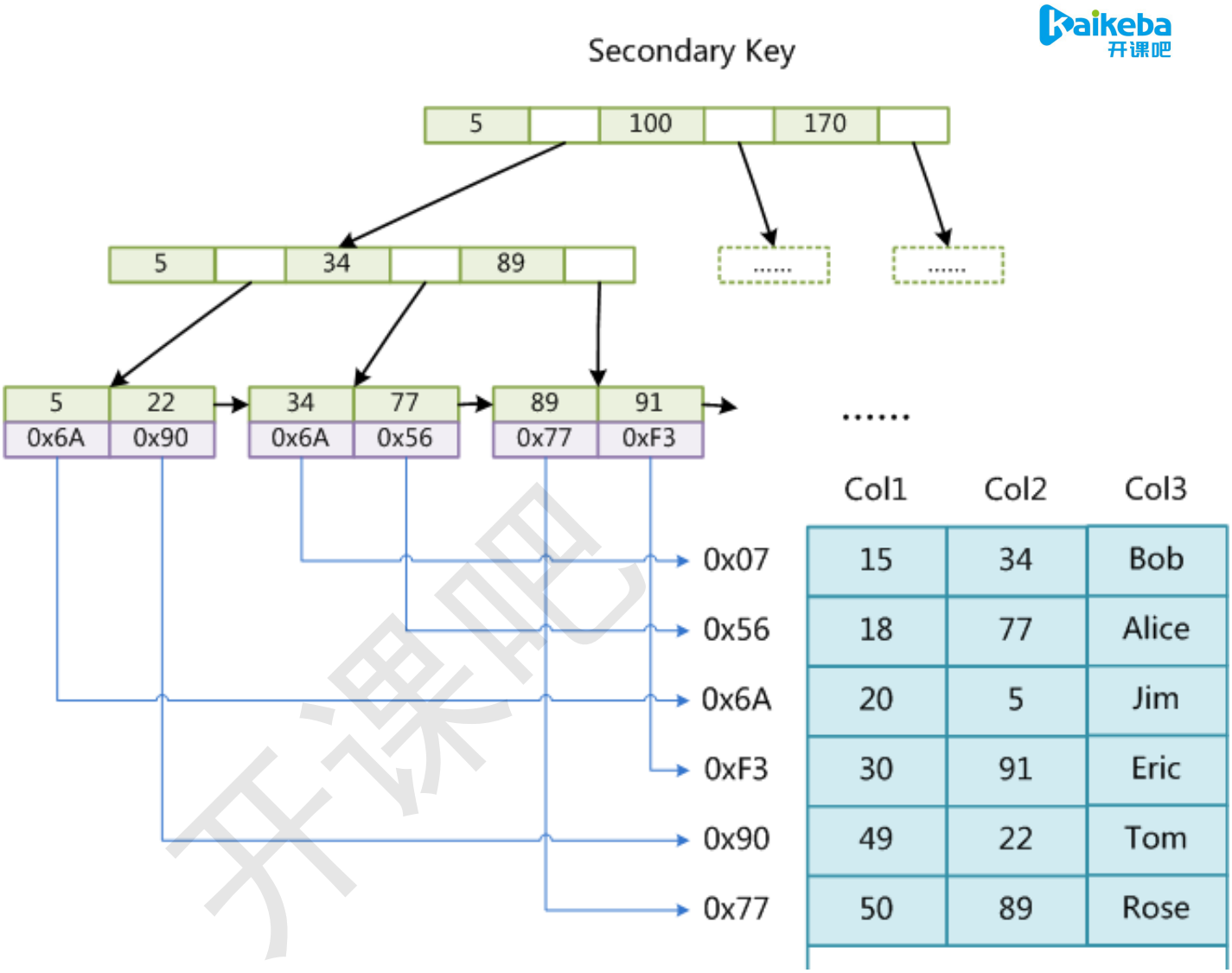
```
CREATE TABLE `tb` (
   `id` int(10) NOT NULL auto_increment,
   `age` int(10) NOT NULL,
   `name` varchar(32) NOT NULL,
    PRIMARY KEY  (`id`),
    KEY `idx_name` (`name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```





```
CREATE TABLE `tb` (
   `id` int(10) NOT NULL auto_increment,
   `age` int(10) NOT NULL,
   `name` varchar(32) NOT NULL,
    PRIMARY KEY  (`id`),
    KEY `idx_age` (`age`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```





3-5、事务

- 原子性（atomicity）
- 一致性（consistency）
- 隔离性（isolation）
- 持久性（durability）

四、实战案例

4-1、设计一个应用商店的数据库

- APP信息表
- APP扩展信息表
- APP分类表
- APP分类关系表
- APP评论表
- 用户安装信息表
- APP评分

4-2、SQL优化

4-2-1、一般的优化原则

一般来说查询性能低下的原因是访问了太多的数据：

- 是否请求了不需要的数据
- 不需要的列不提取，不需要的行不取
- 对索引查找使用where子句消除不匹配的行
- 使用覆盖索引（Extra列是Using Index）避免访问的行
- 从表中检索出数据，过滤掉不匹配的行（Extra列是Using Where）
- 更改架构，例如使用汇总表，将分析的结果进行汇总。
- 将一个复杂的查询分解成多个简单的查询
- 批量处理法
  对于需要处理大量数据的语句，批量进行处理。

4-2-2、索引的优化

- 使用自增ID做PRIMARY KEY,业务主键做UNIQUE KEY
- 一般来说,status,type这类枚举值很少的字段,不适合单独作为索引字段的
- 索引并不是越多越好,无用的索引要删除,冗余的索引
- 不要使用%xxx%这种模糊匹配,会导致全表扫描/索引全扫描

4-2-1、explain的使用 ``` mysql> explain select a, b from t where - +-------------+-------+------------+------+---------------+------+---------+-------+--------+----------+-------------+ | id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra | +----+-------------+-------+------------+------+---------------+------+---------+-------+--------+----------+-------------+ | 1 | SIMPLE | t | NULL | ref | bc | bc | 4 | const | 220622 | 100.00 | Using index | +----+-------------+-------+------------+------+---------------+------+---------+-------+--------+----------+-------------+ 1 row in set, 1 warning (0.03 sec) ```

- id：SQL的id
- select_type：查询的类型
  - SIMPLE(简单SELECT,不使用UNION或子查询等)
  - PRIMARY(查询中若包含任何复杂的子部分,最外层的select被标记为PRIMARY)
  - UNION(UNION中的第二个或后面的SELECT语句)
  - DEPENDENT UNION(UNION中的第二个或后面的SELECT语句，取决于外面的查询)
  - UNION RESULT(UNION的结果)
  - SUBQUERY(子查询中的第一个SELECT)
  - DEPENDENT SUBQUERY(子查询中的第一个SELECT，取决于外面的查询)
  - DERIVED(派生表的SELECT, FROM子句的子查询)
  - UNCACHEABLE SUBQUERY(一个子查询的结果不能被缓存，必须重新评估外链接的第一行)
- table：查询关联的表

- type：访问类型

  - ALL：Full Table Scan， MySQL将遍历全表以找到匹配的行
  - index: Full Index Scan，index与ALL区别为index类型只遍历索引树
  - range:只检索给定范围的行，使用一个索引来选择行
  - ref: 表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值
  - eq_ref: 类似ref，区别就在使用的索引是唯一索引，对于每个索引键值，表中只有一条记录匹配
  - const、system: 当MySQL对查询某部分进行优化，并转换为一个常量时，使用这些类型访问
  - NULL: MySQL在优化过程中分解语句，执行时甚至不用访问表或索引
  - ......

  结果值从好到坏依次是：system > const > eq_ref > ref > range > index > ALL

- possible_keys：可能使用的索引

- key： 实际使用的索引。如果为NULL，则没有使用索引。
- key_len：使用的索引的长度。在不损失精确性的情况下，长度越短越好
- ref：显示索引的哪一列被使用了，如果可能的话，是一个常数
- rows：表示MySQL根据统计信息及索引选用情况估算的找到所需的记录所需要读取的行数
- Extra：关于MYSQL如何解析查询的额外信息
    - Using filesort: MySQL中无法利用索引完成的排序操作
    - Using index：从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。
    - Using temporary：表示MySQL需要使用临时表来存储结果集，常见于排序和分组查询
    - Impossible where：这个值强调了where语句会导致没有符合条件的行
    - Distinct:MySQL发现第1个匹配行后,停止为当前的行组合搜索更多的行。
    - Using where:查询的列未被索引覆盖，where筛选条件非索引的前导列
    - .....

五、其他

5-1、参考资料及推荐阅读

- 官网手册：https://dev.mysql.com/doc/refman/5.6/en/
- 《高性能MySQL》
- 《高可用MySQL》
- 《深入理解MySQL核心技术》
- MySQL的源码

5-2、课后练习

> 设计一个直播系统（例如斗鱼直播）的数据库