MySQL逻辑分层

1. 连接层
2. 服务层
3. 引擎层

InnoDB ：事务优先 （适合高并发操作：行锁） mysql 默认支持

MyISAM ：性能优先 （表锁）

4、存储层

查询数据库支持引擎

show engines;

查看当前使用的引擎

show variables like '%storage\_engine%';

指定数据库对象的引擎：

SQL优化

原因：性能较低、执行时间太长，等待时间过长，SQL语句欠佳（连接查询）、索引失效、服务器参数设置不合理

1. SQL：

Select …from..join..on..where..groud by….having…order by..limit

解析过程：

From on join where…groud by having select order by limit

1. SQL优化， 主要就是在优化索引

索引：相当于书的目录 (index)是帮助mysql高效获取数据的数据结构 （树：B树，hash树）

三层B树可以存放上百万条数据 一般指B+数

一、索引的弊端：

1. 索引本身很大，可以存在内存、硬盘中（一般在）
2. 索引不是所有的情况都适用：1、少量数据 2、频繁更新的数据 3、很少使用的字段
3. 索引可以提高查询的效率，但是会降低增删改的效率

二、索引的优势：

1. 提高查询效率（降低IO使用率）
2. 降低CPU使用率

create table tb(

id int(4) auto\_increment,

name varchar(5),

dept varchar(5),

primary key(id)

)ENGINE=MyISAM AUTO\_INCREMENT=1

DEFAULT CHARSET=utf8

三、索引：

分类：

单值索引：单列，单个字段的索引 一张表可以有多个单值索引

主键索引：不能重复 一般用 id 不能是null

(一个字段是primary key,则该字段默认就是主键索引)

唯一索引：不能重复 一般用 id 可以是null

复合索引：多个列构成的索引 （相当于二级索引）

创建索引：

方式一：

create 索引类型 索引名 on 表（字段）

单值：

create index dept\_index on tb(dept);

唯一：

create unique index name\_index on tb(name);

复合索引：

create index dept\_name\_index on tb(dept,name);

方式二：

alter table 表名 索引类型 索引名（字段）

删除索引：

drop index 索引名 on 表名；

eg : drop index name\_idnex on tb;

查询索引：

show index from 表名；

eg : show index from tb;

四、SQL 性能问题

A、分析SQL的执行计划 ：explain ，可以模拟SQL优化器执行SQL语句，从而让开 发人员看到自己的SQL状况

B、mysql查询优化器会干扰我们的优化

优化方法，官网：<https://dev.mysql.com/doc/refman/5.7/en/optimization.html>

查询执行计划： explain + SQL语句

explain select \* from tb;

id: 1 编号

select\_type: SIMPLE 查询类型

table: NULL

partitions: NULL

type: NULL

possible\_keys: NULL 预测的使用的索引

key: NULL 实际使用的索引

key\_len: NULL 实际使用的长度

ref: NULL 表之间的引用

rows: NULL 优化器预估的记录扫描行数

filtered: NULL

Extra: no matching row in const table 额外的信息

准备数据

create table course(

cid int(3),

cname varchar(20),

tid int(3)

);

create table teacher(

tid int(3),

tname varchar(20),

tcid int(3)

);

create table teacherCard(

tcid int(3),

tcdesc varchar(200)

);

insert into course values(1,'java',1);

insert into course values(2,'html',1);

insert into course values(3,'sql',2);

insert into course values(4,'web',3);

insert into teacher values(1,'tz',1);

insert into teacher values(2,'tw',2);

insert into teacher values(3,'tl',3);

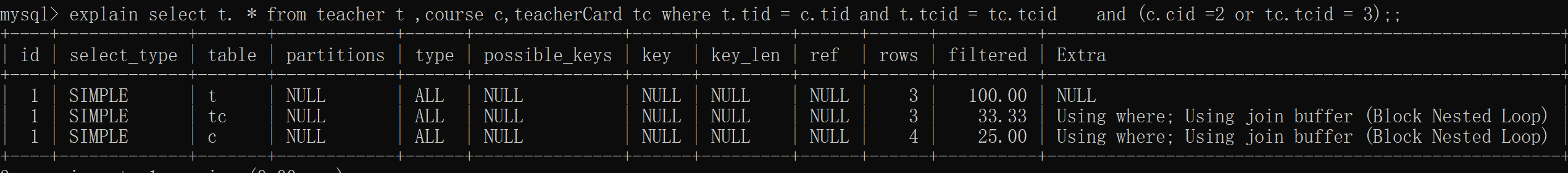
insert into teacherCard values(1,'tzdesc');

insert into teacherCard values(2,'twdesc');

insert into teacherCard values(3,'tldesc');

eg : 查询课程为2 或 教师证编号为3 的老师信息

select t. \* from teacher t ,course c,teacherCard tc where t.tid = c.tid and t.tcid = tc.tcid and (c.cid =2 or tc.tcid = 3);



id: 1 id值相同，从上向下顺序执行 t-tc-c 表的数据越多靠后 、

表的执行顺序 因数量的改变而改变的原因 笛卡尔积

select\_type: SIMPLE

PRIMARY: 包含子查询SQL中的 主查询 （最外层）

SUBQUERY: 包含子查询SQL中的 子查询 （非最外层）

simple: 简单查询（不包含子查询，union）

derived:衍生查询（使用到了临时表）

table: NULL

partitions: NULL

type: NULL

possible\_keys: NULL 预测的使用的索引

key: NULL 实际使用的索引

key\_len: NULL 实际使用的长度

ref: NULL 表之间的引用

rows: NULL 优化器预估的记录扫描行数

filtered: NULL

Extra: no matching row in const table 额外的信息

Id值不同 ：id值越大越优先查询（本质：在嵌套子查询时，先查内层 再查外层）

Id值相同有不同：id值越大越优先执行，id值相同从上往下顺序执行