数据库优化

ju7ran

目　录

1-数据库-基本使用

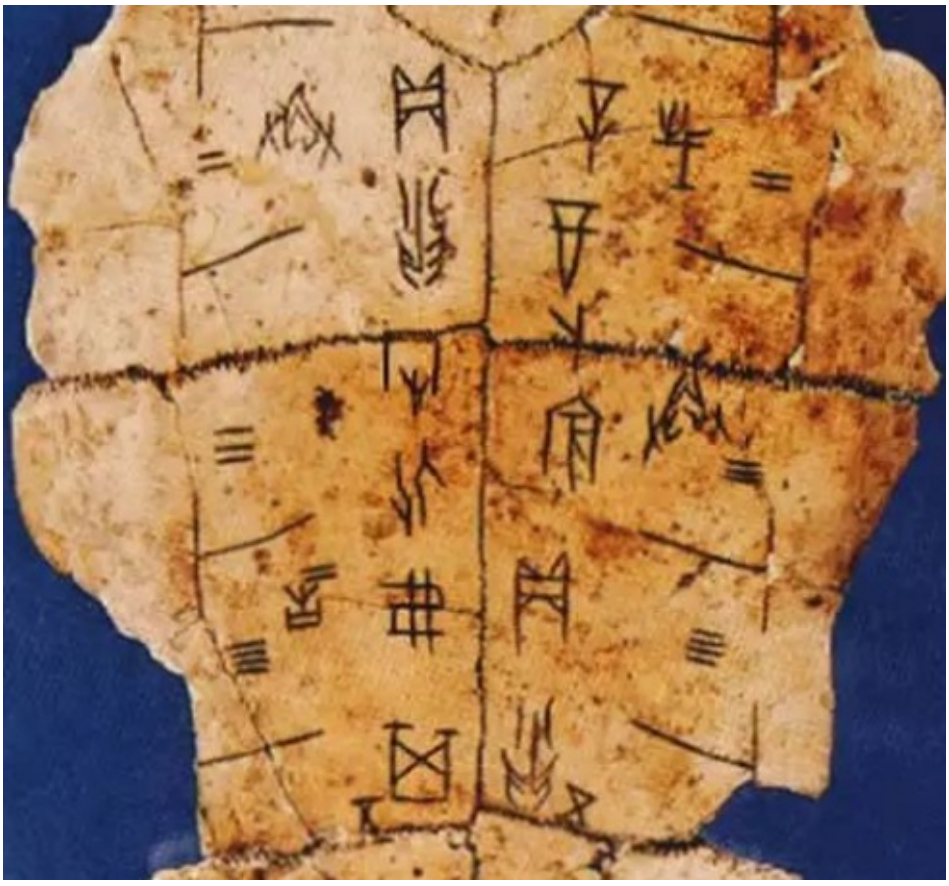
1-1-数据存储

逻辑教育
Logic education

数据库的介绍

数据存储

以前是这样记录的



传统记录数据的缺点:

- 不易保存
- 备份困难
- 查找不便

现代化手段----文件

- 对于数据容量较大的数据，不能够很好的满足，而且性能较差
- 不易扩展

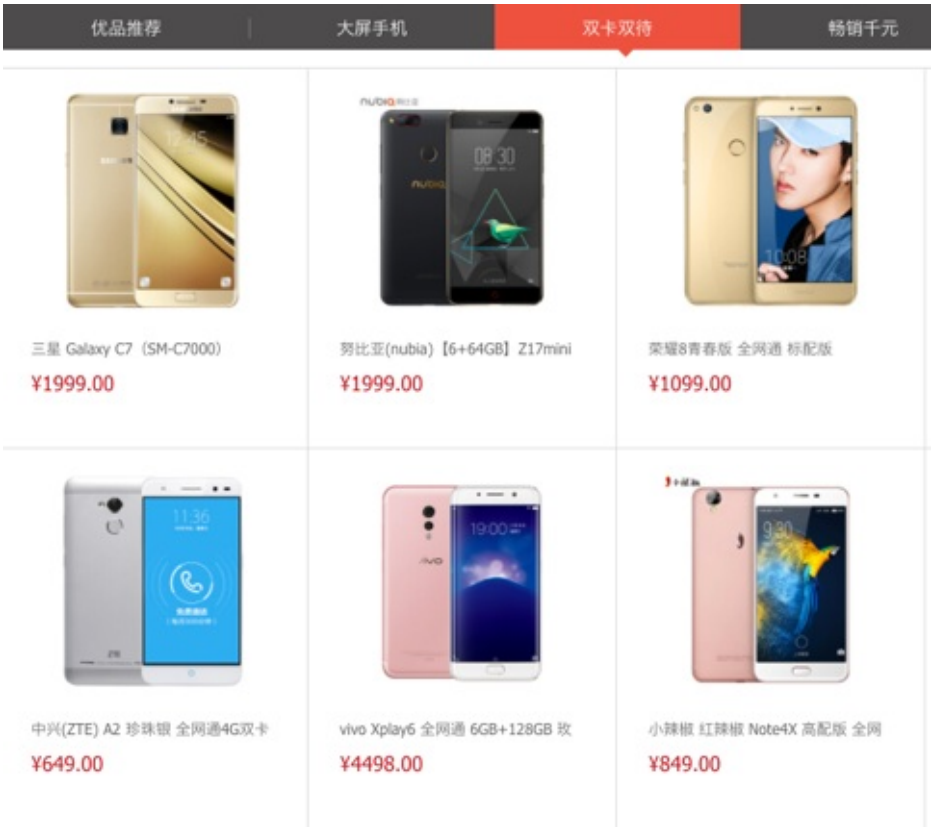
数据库

- 持久化存储
- 读写速度极高
- 保证数据的有效性
- 对程序支持性非常好，容易扩展

我们将来看到的是这个样子的

id	name	age	height	gender	cls_id	isdelete
1	juran	1	180	1	1	0
2	juran2	6	175.5	0	2	0
3	居然1	6	(NULL)	1	1	1
4	居然	12	(NULL)	0	2	0
5	567	9	(NULL)	1	2	0
6	34534	11	(NULL)	1	1	0
7	564756	13	(NULL)	1	1	1
8	345645	1	(NULL)	1	2	0

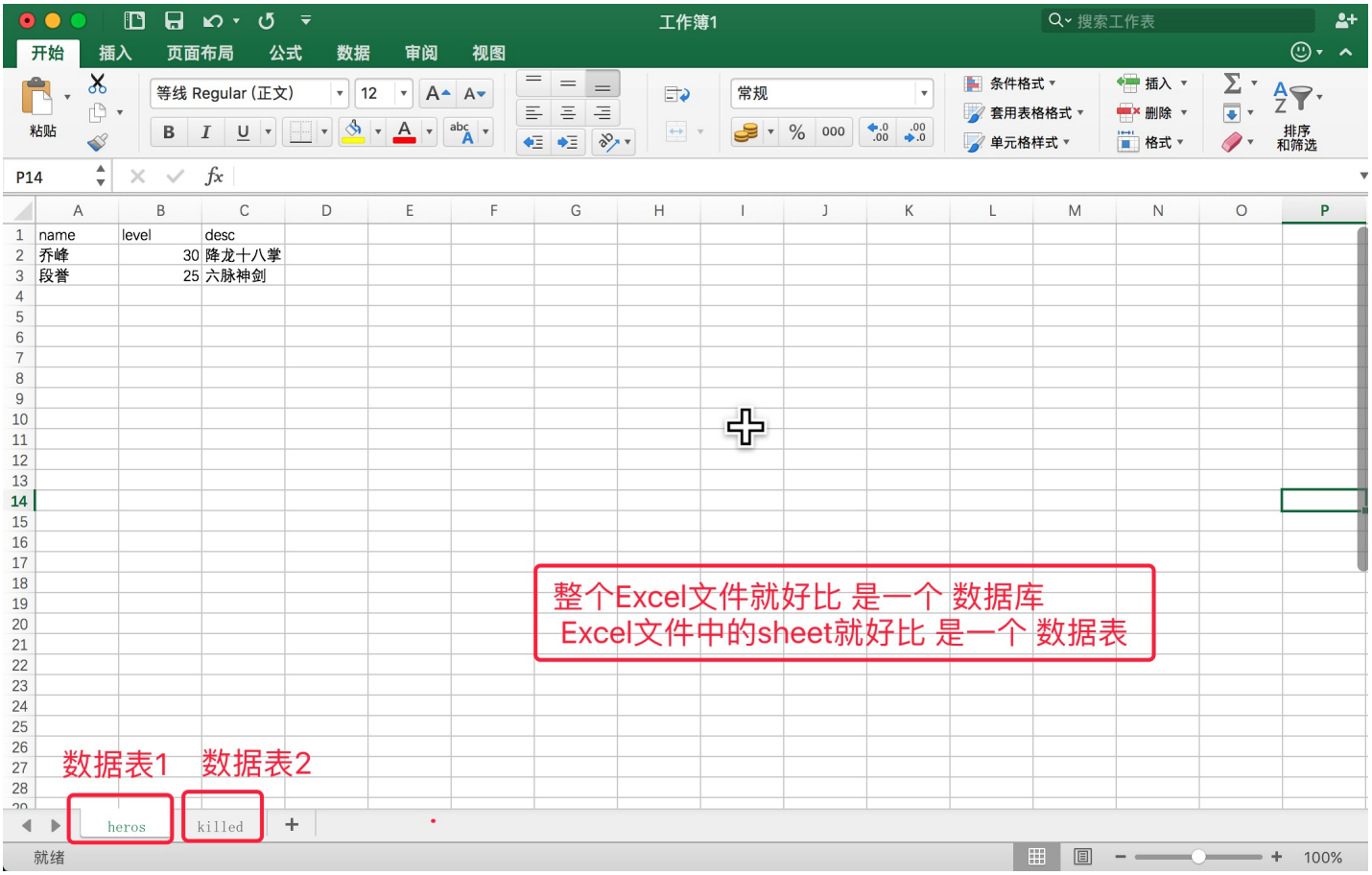
实际上在网页上展示出来的是



1-2-数据库

理解数据库

- 数据行(记录)
- 数据列(字段)
- 数据表(数据行的集合)
- 数据库(数据表的集合)



MySQL 简介

MySQL是一个关系型数据库管理系统，由瑞典MySQL AB公司开发，后来被Sun公司收购，Sun公司后来又被

Oracle公司收购，目前属于Oracle旗下产品

特点

- 使用C和C++编写，并使用了多种编译器进行测试，保证源代码的可移植性
- 支持多种操作系统，如Linux、Windows、AIX、FreeBSD、HP-UX、MacOS、NovellNetware、OpenBSD、OS/2 Wrap、Solaris等
- 为多种编程语言提供了API，如C、C++、Python、Java、Perl、PHP、Eiffel、Ruby等
- 支持多线程，充分利用CPU资源
- 优化的SQL查询算法，有效地提高查询速度
- 提供多语言支持，常见的编码如GB2312、BIG5、UTF8
- 提供TCP/IP、ODBC和JDBC等多种数据库连接途径
- 提供用于管理、检查、优化数据库操作的管理工具
- 大型的数据库。可以处理拥有上千万条记录的大型数据库
- 支持多种存储引擎
- MySQL 软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择MySQL作为网站数据库
- MySQL使用标准的SQL数据语言形式
- Mysql是可以定制的，采用了GPL协议，你可以修改源码来开发自己的Mysql系统
- 在线DDL更改功能
- 复制全局事务标识
- 复制无崩溃从机
- 复制多线程从机

1-3-MySQL安装和配置

安装和配置

直接安装

下载地址：www.mysql.com/downloads

安装：https://jingyan.baidu.com/article/0aa223751ed91188cc0d643f.html

集成安装

- phpstudy
- xampp

图形化管理工具

- 1.phpMyAdmin
- 2.Navicat
- 3.SQLyog

1-4-SQL

SQL

SQL是结构化查询语言，是一种用来操作RDBMS(关系型数据库管理系统)的数据库语言，当前关系型数据库都支持使用SQL语言进行操作，也就是说可以通过SQL操作oracle，sql server,mysql等关系型数据库。

SQL语句主要分为

- DQL：数据查询语言，用于对数据进行查询，如select
- DML：数据操作语言，对数据进行增加、修改、删除，如insert、udpate、delete
- DDL：数据定义语言，进行数据库、表的管理等，如create、drop

重点是数据的crud（增删改查），必须熟练编写DQL、DML，能够编写DDL完成数据库、表的操作

```
# 创建Connection连接
conn = connect(host='localhost', port=3306, user='root', password='mysql', data
base='python1', charset='utf8')
# 得Cursor对象
cs = conn.cursor()
# 更新
# sql = 'update students set name="刘邦" where id=6'
# 删除
# sql = 'delete from students where id=6'
# 执行select语句，并返回受影响的行数：查询一条学生数据
sql = 'select id,name from students where id = 7'
# sql = 'SELECT id,name FROM students WHERE id = 7'
count=cs.execute(sql)
# 打印受影响的行数
print(count)
```

1-5-数据完整性

数据完整性

在表中为了更加准确的存储数据，保证数据的正确有效，可以在创建表的时候，为表添加一些强制性的验证，包括数据字段的类型、约束

常见的数据类型

- 整数：int
- 小数：decimal
- 字符串：varchar,char
- 日期时间: date, time, datetime
- 枚举类型:enum

特别说明的类型

decimal表示浮点数，如decimal(5,2)表示共存5位数，小数占2位

char表示固定长度的字符串，如char(3)，如果填充'ab'时会补一个空格为'ab '

varchar表示可变长度的字符串，如varchar(3)，填充'ab'时就会存储'ab'

字符串text表示存储大文本，当字符大于4000时推荐使用

对于图片、音频、视频等文件，不存储在数据库中，而是上传到某个服务器上，然后在表中存储这个文件的保存路径

更全的数据类型可以参考：http://blog.csdn.net/anxpp/article/details/51284106

数值类型

类型	字节大小	有符号范围(Signed)	无符号范围(Unsigned)
TINYINT	1	-128 ~ 127	0 ~ 255
SMALLINT	2	-32768 ~ 32767	0 ~ 65535
MEDIUMINT	3	-8388608 ~ 8388607	0 ~ 16777215
INT/INTEGER	4	-2147483648 ~2147483647	0 ~ 4294967295
BIGINT	8	-9223372036854775808 ~ 9223372036854775807	0 ~ 18446744073709551615

字符串

类型	字节大小	示例
CHAR	0-255	类型:char(3) 输入 'ab', 实际存储为'ab ', 输入'abcd' 实际存储为 'abc'
VARCHAR	0-255	类型:varchar(3) 输 'ab',实际存储为'ab', 输入'abcd',实际存储为'abc'
TEXT	0-65535	大文本

日期时间类型

类型	字节大小	示例
DATE	4	'2020-01-01'
TIME	3	'12:29:59'
DATETIME	8	'2020-01-01 12:29:59'
YEAR	1	'2017'
TIMESTAMP	4	'1970-01-01 00:00:01' UTC ~ '2038-01-01 00:00:01' UTC

约束

- 主键primary key：物理上存储的顺序
- 非空not null：此字段不允许填写空值
- 惟一unique：此字段的值不允许重复
- 默认default：当不填写此值时会使用默认值，如果填写时以填写为准
- 外键foreign key：对关系字段进行约束，当为关系字段填写值时，会到关联的表中查询此值是否存在，如果存在则填写成功，如果不存在则填写失败并抛出异常

1-6-命令行操作数据库

数据库操作

- 数据库的连接
- 退出数据库
- 查看所有数据库
- 显示数据库版本
- 显示时间
- 创建数据库
- 查看创建数据库的命令
- 查看已经创建的数据库
- 查看当前使用的数据库
- 删除数据库
- 使用数据库

数据表的操作

- 查看当前数据库中所有表
- 创建表
- 查看表
- 查看表的创建语句
- 修改表-添加字段
- 修改表-修改字段：不重命名版
- 修改表-修改字段：重命名版
- 修改表-删除字段
- 删除表或者数据库
- 查看表的创建语句

对数据进行管理

- 新增
- 修改
- 删除
- 查询

2-MySQL查询

2-1-MySQL查询

MySQL查询

select基础语法

```
select * from 表名字;
```

select完整语法

```
select 去重选项 字段列表 [as 字段别名] from 数据源 [where子句] [group by 子句] [having子句] [order by 子句] [limit子句];
```

- 查询所有字段
- 查询指定字段
- 使用 as 给字段起别名
- 可以通过 as 给表起别名

消除重复行

- 在select后面列前使用distinct可以消除重复的行

2-2-条件

条件

使用where子句对表中的数据筛选，结果为true的行会出现在结果集中

语法

```
select * from 表名 where 条件;
例：
select * from students where id=1;
```

where后面支持多种运算符，进行条件的处理

- 比较运算符
- 逻辑运算符
- 模糊查询
- 范围查询
- 空判断

比较运算符

- 等于: =
- 大于: >
- 大于等于: >=
- 小于: <
- 小于等于: <=
- 不等于: != 或 <>

逻辑运算符

- and

- or
- not

模糊查询

- like
- %表示任意多个任意字符
- _表示一个任意字符

范围查询

- in表示在一个非连续的范围内
- between ... and ...表示在一个连续的范围内

空判断

- 注意:null(占用空间地址的)与""(空字符串,不占空间地址的)是不用的
- 判断空 `is null` ,这里注意不能用等于(=)
- 判断非空 `is not null`

2-3-聚合函数

聚合函数

总数

- count(*)表示计算总行数，括号中写星与列名，结果是相同的
- max(列)表示求此列的最大值
- min(列)表示求此列的最小值
- sum(列)表示求此列的和
- avg(列)表示求此列的平均值

2-4-分组

分组

group by

1. group by的含义:将查询结果按照1个或多个字段进行分组，字段值相同的为一组
2. group by可用于单个字段分组，也可用于多个字段分组

group by + group_concat()

1. group_concat(字段名)可以作为一个输出字段来使用，
2. 表示分组之后，根据分组结果，使用group_concat()来放置每一组的某字段的值的集合

group by + 集合函数

通过group_concat()的启发，我们既然可以统计出每个分组的某字段的值的集合，那么我们也可以通过集合函数来对这个 `值的集合` 做一些操作

group by + having

1. having 条件表达式：用来分组查询后指定一些条件来输出查询结果
2. having作用和where一样，但having只能用于group by

2-5-排序

排序

为了方便查看数据，可以对数据进行排序

语法：

```
select * from 表名 order by 列1 asc|desc [,列2 asc|desc,...]
```

说明

- 将行数据按照列1进行排序，如果某些行列1的值相同时，则按照列2排序，以此类推
- 默认按照列值从小到大排列（asc）
- asc从小到大排列，即升序
- desc从大到小排序，即降序

2-6-分页

获取部分行

当数据量过大时，在一页中查看数据是一件非常麻烦的事情

语法

```
select * from 表名 limit start,count
```

说明

- 从start开始，获取count条数据

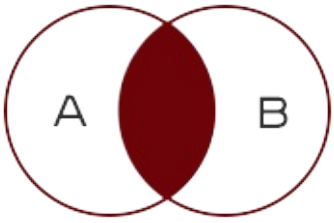
2-7-连接查询

连接查询

当查询结果的列来源于多张表时，需要将多张表连接成一个大的数据集，再选择合适的列返回

mysql支持三种类型的连接查询，分别为：

- 内连接查询：查询的结果为两个表匹配到的数据



- 左连接查询：查询的结果为两个表匹配到的数据，左表特有的数据，对于右表中不存在的数据使用null填充

- 右连接查询：查询的结果为两个表匹配到的数据，右表特有的数据，对于左表中不存在的数据使用null填充

语法

```
select * from 表1 inner或left或right join 表2 on 表1.列 = 表2.列
```

2-8-子查询

子查询

在一个 select 语句中,嵌入了另外一个 select 语句, 那么被嵌入的 select 语句称之为子查询语句

语法

查询最高的男生信息

```
select * from students where height = (select max(height) from students);
```

2-9-自关联

自关联

- 设计省信息的表结构provinces
  - id
  - ptitle
- 设计市信息的表结构citys
  - id
  - ctitle
  - proid
- citys表的proid表示城市所属的省，对应着provinces表的id值

问题：

> 能不能将两个表合成一张表呢？

思考：

> 观察两张表发现，citys表比provinces表多一个列proid，其它列的类型都是一样的

意义：

> 存储的都是地区信息，而且每种信息的数据量有限，没必要增加一个新表，或者将来还要存储区、乡镇信息，都增加新表的开销太大

答案：

> 定义表areas，结构如下
>
> - id
> - atitle

> - pid

3-MySQL外键

MySQL外键

在实际开发的项目中，一个健壮数据库中的数据一定有很好的参照完整性。例如学生表和成绩单两张表，如果成绩单中有张三的成绩，学生表中张三的档案却被删除了，这样就会产生垃圾数据或者错误数据。为了保证数据的完整性，将两张表之间的数据建立关系，因此就需要在成绩表中添加外键约束。

班级表

```
create table classes(
    id int(4) not null primary key,
    name varchar(36)
);
```

学生表

```
create table student(
    sid int(4) not null primary key,
    sname varchar(36),
    gid int(4) not null
);
```

引入外键之后，外键列只能插入参照列存在的值，参照列被参照的值不能被删除，这就保证了数据的参照完整性。

添加外键

语法

```
alert table 表名 add constraint 外键名字 foreign key(外键字段名) references 外表表名(主键字段名)
```

验证外键的作用

```
insert into student (sid, sname, gid) values(1000, 'juran', 123);
```

删除外键约束

```
alter table 表名 drop foreign key 外键名;
```

4-MySQL与Python交互

4-1-数据准备

准备数据

创建数据表

```
-- 创建 "京东" 数据库
create database jd charset=utf8;

-- 使用 "京东" 数据库
use  jd;

-- 创建一个商品goods数据表
create table goods(
    id int unsigned primary key auto_increment not null,
    name varchar(150) not null,
    cate_name varchar(40) not null,
    brand_name varchar(40) not null,
    price decimal(10,3) not null default 0,
    is_show bit not null default 1,
    is_saleoff bit not null default 0
);
```

插入数据

```
-- 向goods表中插入数据

insert into goods values(0,'r510vc 15.6英寸笔记本','笔记本','华硕','3399',default,
default);
insert into goods values(0,'y400n 14.0英寸笔记本电脑','笔记本','联想','4999',default
,default);
insert into goods values(0,'g150th 15.6英寸游戏本','游戏本','雷神','8499',default,
default);
insert into goods values(0,'x550cc 15.6英寸笔记本','笔记本','华硕','2799',default,
default);
insert into goods values(0,'x240 超极本','超级本','联想','4880',default,default);
```

```
insert into goods values(0,'u330p 13.3英寸超极本','超级本','联想','4299',default,default);
insert into goods values(0,'svp13226scb 触控超极本','超级本','索尼','7999',default,default);
insert into goods values(0,'ipad mini 7.9英寸平板电脑','平板电脑','苹果','1998',default,default);
insert into goods values(0,'ipad air 9.7英寸平板电脑','平板电脑','苹果','3388',default,default);
insert into goods values(0,'ipad mini 配备 retina 显示屏','平板电脑','苹果','2788',default,default);
insert into goods values(0,'ideacentre c340 20英寸一体电脑 ','台式机','联想','3499',default,default);
insert into goods values(0,'vostro 3800-r1206 台式电脑','台式机','戴尔','2899',default,default);
insert into goods values(0,'imac me086ch/a 21.5英寸一体电脑','台式机','苹果','9188',default,default);
insert into goods values(0,'at7-7414lp 台式电脑 linux ）','台式机','宏碁','3699',default,default);
insert into goods values(0,'z220sff f4f06pa工作站','服务器/工作站','惠普','4288',default,default);
insert into goods values(0,'poweredge ii服务器','服务器/工作站','戴尔','5388',default,default);
insert into goods values(0,'mac pro专业级台式电脑','服务器/工作站','苹果','28888',default,default);
insert into goods values(0,'hmz-t3w 头戴显示设备','笔记本配件','索尼','6999',default,default);
insert into goods values(0,'商务双肩背包','笔记本配件','索尼','99',default,default);

insert into goods values(0,'x3250 m4机架式服务器','服务器/工作站','ibm','6888',default,default);
insert into goods values(0,'商务双肩背包','笔记本配件','索尼','99',default,default);
```

4-2-数据表的拆分

创建 "商品分类" 表

```
create table goods_cates(
    id int unsigned primary key auto_increment not null,
    name varchar(40) not null
);
```

- 查询goods表中商品的种类

```
select cate_name from goods group by cate_name;
```

- 将分组结果写入到goods_cates数据表

```
insert into goods_cates (name) select cate_name from goods group by cate_name;
```

同步表数据

- 通过goods_cates数据表来更新goods表

```
update goods as g inner join goods_cates as c on g.cate_name=c.name set g.cate_
name=c.id;
```

4-3-Python操作MySQL

Python-mysql安装

安装pymysql

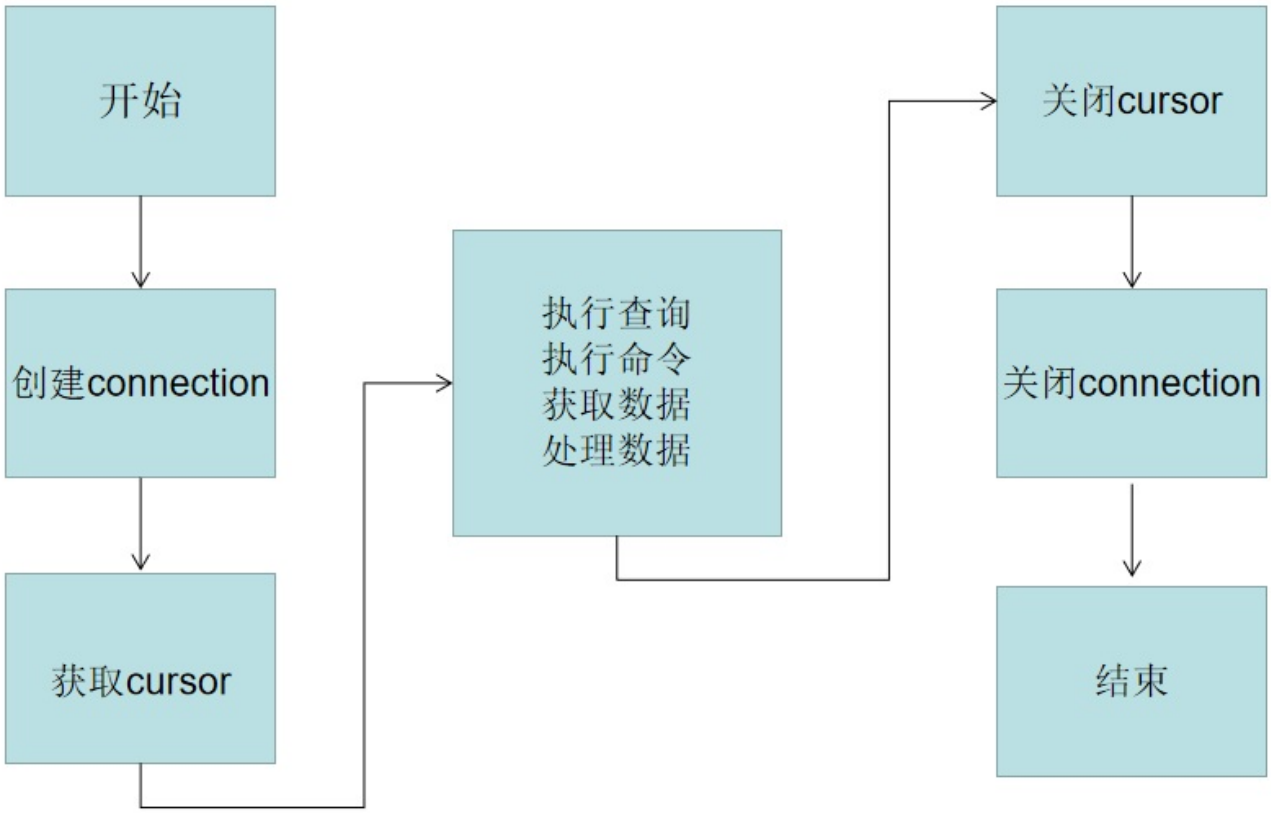
在Windows操作系统上安装

Python3: `pip install pymysql`

Python2: `pip install MySQLdb`

Ubuntu安装：https://www.jianshu.com/p/d84cdb5e6273

Python操作MySQL步骤



Connection 对象

用于建立与数据库的连接

创建对象：调用connect()方法

```
conn=connect(参数列表)

参数host：连接的mysql主机，如果本机是'localhost'
参数port：连接的mysql主机的端口，默认是3306
参数database：数据库的名称
参数user：连接的用户名
参数password：连接的密码
参数charset：通信采用的编码方式，推荐使用utf8

import pymysql

con = pymysql.connect(host = 'localhost',port=3306,database='python-01',user='root',password = 'root',charset = 'utf8')

from pymysql import *

conn = connect(host = 'localhost',port=3306,database='python-01',user='root',password = 'root',charset = 'utf8')
```

对象的方法

- close()关闭连接
- commit()提交
- cursor()返回Cursor对象,用于执行sql语句并获得结果

Cursor对象

- 用于执行sql语句,使用频度最高的语句为select、insert、update、delete
- 获取Cursor对象:调用Connection对象的cursor()方法

```
cs1=conn.cursor()
```

对象的方法

- close()关闭 先关闭游标,在关闭链接
- execute(operation [, parameters])执行语句，返回受影响的行数，主要用于执行insert、update、delete语句，也可以执行create、alter、drop等语句
- fetchone()执行查询语句时，获取查询结果集的第一个行数据，返回一个元组
- fetchall()执行查询时，获取结果集的所有行，一行构成一个元组，再将这些元组装入一个元组返回

使用python连接数据库

```
from pymysql import *

try:
    conn = connect(
        host = "localhost",
        port = 3306,
        user = "root",
        passwd = "root",
        db = 'logic_web',
        charset = 'utf8'
    )
    cursor = conn.cursor()
    cursor.execute('select * from users_banner')
    result = cursor.fetchone()
    cursor.close()
    conn.close()
except Exception as e:
    print("Error %d:%s"%(e.args[0],e.args[1]))
```

代码实现查询数据库中的数据

练习商品查询

5-MySQL高级

5-1-视图

视图

1. 问题

对于复杂的查询，往往是有多个数据表进行关联查询而得到，如果数据库因为需求等原因发生了改变，为了保证查询出来的数据与之前相同，则需要在多个地方进行修改，维护起来非常麻烦

解决办法：定义视图

2. 视图是什么

通俗的讲，视图就是一条SELECT语句执行后返回的结果集。所以我们在创建视图的时候，主要的工作就落在创建这条SQL查询语句上。

视图是对若干张基本表的引用，一张虚表，查询语句执行的结果，不存储具体的数据（基本表数据发生了改变，视图也会跟着改变）；

方便操作，特别是查询操作，减少复杂的SQL语句，增强可读性；

3. 定义视图

建议以v_开头

```
create view 视图名称 as select语句;
```

4. 查看视图

查看表会将所有的视图也列出来

```
show tables;
```

5. 使用视图

视图的用途就是查询

```
select * from v_stu_score;
```

6. 删除视图

```
drop view 视图名称;
例：
drop view v_stu_sco;
```

7.视图案例

```
select p.id,p.`province`,c.`city` from provinces as p inner join cities as c on
 p.`provinceid` = c.provinceid having p.`province` = '黑龙江省';
```

8.视图的修改

有下列内容之一，视图不能做修改

- select子句中包含distinct
- select子句中包含组函数
- select语句中包含group by子句
- select语句中包含order by子句
- where子句中包含相关子查询
- from子句中包含多个表
- 如果视图中有计算列，则不能更新
- 如果基表中有某个具有非空约束的列未出现在视图定义中，则不能做insert操作

9. 视图的作用

1. 提高了重用性，就像一个函数
2. 对数据库重构，却不影响程序的运行
3. 提高了安全性能，可以对不同的用户
4. 让数据更加清晰

5-2-事务

事务

1. 为什么要有事务

事务广泛的运用于订单系统、银行系统等多种场景
例如：

> A用户和B用户是银行的储户，现在A要给B转账500元，那么需要做以下几件事：
>
> 1. 检查A的账户余额>500元；
> 2. A 账户中扣除500元;
> 3. B 账户中增加500元;

正常的流程走下来，A账户扣了500，B账户加了500，皆大欢喜。

那如果A账户扣了钱之后，系统出故障了呢？A白白损失了500，而B也没有收到本该属于他的500。

以上的案例中，隐藏着一个前提条件：A扣钱和B加钱，要么同时成功，要么同时失败。事务的需求就在于此

事务

所谓事务,它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位

例如，银行转帐工作：从一个帐号扣款并使另一个帐号增款，这两个操作要么都执行，要么都不执行。所以，应该把他们看成一个事务。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性

假如一个银行的数据库有两张表：支票表（checking）和储蓄表（savings）。现在要从用户Jane的支票账户转移200美元到她的储蓄账户，那么至少需要三个步骤：

1.检查支票账户的余额高于或者等于200美元。

2.从支票账户余额中减去200美元。

3.在储蓄帐户余额中增加200美元。

上述三个步骤的操作必须打包在一个事务中，任何一个步骤失败，则必须回滚所有的步骤。

事务四大特性(简称ACID)

- 原子性(Atomicity)
- 一致性(Consistency)
- 隔离性(Isolation)
- 持久性(Durability)

- 原子性（ atomicity ）

> 一个事务必须被视为一个不可分割的最小工作单元，整个事务中的所有操作要么全部提交成功，要么全部失败回滚，对于一个事务来说，不可能只执行其中的一部分操作，这就是事务的原子性

- 一致性（ consistency ）

> 数据库总是从一个一致性的状态转换到另一个一致性的状态。（ 在前面的例子中，一致性确保了，即使在执行第三、四条语句之间时系统崩溃，支票账户中也不会损失200美元，因为事务最终没有提交，所以事务中所做的修改也不会保存到数据库中。 ）

- 隔离性（ isolation ）

> 通常来说，一个事务所做的修改在最终提交以前，对其他事务是不可见的。（ 在前面的例子中，当执行完第三条语句、第四条语句还未开始时，此时有另外的一个账户汇总程序开始运行，则其看到支票帐户的余额并没有被减去200美元。 ）

- 持久性（ durability ）

> 一旦事务提交，则其所做的修改会永久保存到数据库。（ 此时即使系统崩溃，修改的数据也不会丢失。 ）

事务命令

表的引擎类型必须是innodb类型才可以使用事务，这是mysql表的默认引擎

开启事务，命令如下：

- 开启事务后执行修改命令，变更会维护到本地缓存中，而不维护到物理表中

```
begin;
或者
start transaction;
```

提交事务，命令如下

- 将缓存中的数据变更维护到物理表中

```
commit;
```

回滚事务，命令如下：

- 放弃缓存中变更的数据

```
rollback;
```

注意

1. 修改数据的命令会自动的触发事务，包括insert、update、delete
2. 而在SQL语句中有手动开启事务的原因是：可以进行多次数据的修改，如果成功一起成功，否则一起会滚到之前的数据

事务练习-模拟银行转账

5-3-索引

索引

思考

> 在图书馆中是如何找到一本书的？

一般的应用系统对比数据库的读写比例在10:1左右(即有10次查询操作时有1次写的操作)，
而且插入操作和更新操作很少出现性能问题，
遇到最多、最容易出问题还是一些复杂的查询操作，所以查询语句的优化显然是重中之重

解决办法

当数据库中数据量很大时，查找数据会变得很慢
优化方案：索引

索引是什么

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度

索引的目的

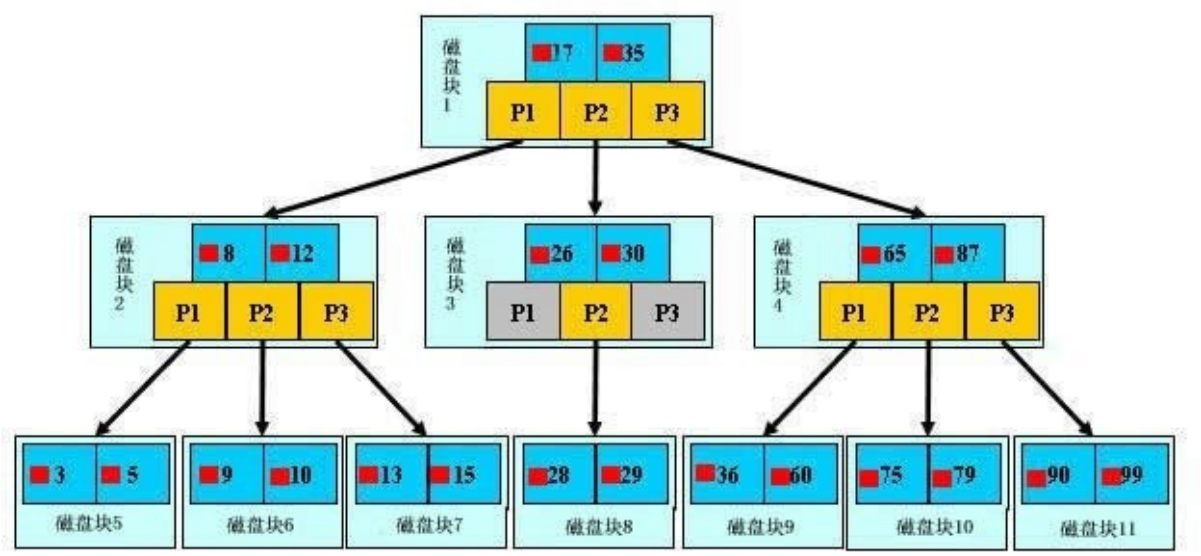
索引的目的在于提高查询效率，可以类比字典，如果要查“mysql”这个单词，我们肯定需要定位到m字母，然后从下往下找到y字母，再找到剩下的sql。

索引原理

除了词典，生活中随处可见索引的例子，如火车站的车次表、图书的目录等。它们的原理都是一样的，通过不断的缩小想要获得数据的范围来筛选出最终想要的结果，同时把随机的事件变成顺序的事件，也就是我们总是通过

同一种查找方式来锁定数据。

数据库也是一样，但显然要复杂许多，因为不仅面临着等值查询，还有范围查询(>、<、between、in)、模糊查询(like)、并集查询(or)等等。数据库应该选择怎么样的方式来应对所有的问题呢？



索引的使用

- 查看索引

```
show index from 表名;
```

- 创建索引
  - 如果指定字段是字符串，需要指定长度，建议长度与定义字段时的长度一致
  - 字段类型如果不是字符串，可以不填写长度部分

```
create index 索引名称 on 表名(字段名称(长度))
```

- 删除索引：

```
drop index 索引名称 on 表名;
```

索引案例

创建测试表test

```
create table test(title varchar(10));
```

- 使用python程序向表中加入十万条数据

查询

- 开启运行时间监测：

```
set profiling=1;
```

- 查找第1万条数据ha-99999

```
select * from test where title='ha-99999';
```

- 查看执行的时间：

```
show profiles;
```

- 为表title_index的title列创建索引：

```
create index title_index on test(title(10));
```

- 执行查询语句：

```
select * from test where title='ha-99999';
```

- 再次查看执行的时间

```
show profiles;
```

适合建立索引的情况

1.主键自动建立索引

2.频繁作为查询条件的字段应该建立索引

3.查询中与其他表关联的字段,外键关系建立索引

4.在高并发的情况下创建复合索引

5.查询中排序的字段,排序字段若通过索引去访问将大大提高排序速度 (建立索引的顺序跟排序的顺序保持一致)

不适合建立索引的情况

频繁更新的字段不适合建立索引

where条件里面用不到的字段不创建索引

表记录太少,当表中数据量超过三百万条数据,可以考虑建立索引

数据重复且平均的表字段,比如性别,国籍

5-4-账户管理(了解)

账户管理

- 在生产环境下操作数据库时，绝对不可以使用root账户连接，而是创建特定的账户，授予这个账户特定的操作权限，然后连接进行操作，主要的操作就是数据的crud
- MySQL账户体系：根据账户所具有的权限的不同，MySQL的账户可以分为以下几种
  - 服务实例级账号：，启动了一个mysqld，即为一个数据库实例；如果某用户如root,拥有服务实例级分配的权限，那么该账号就可以删除所有的数据库、连同这些库中的表
  - 数据库级别账号：对特定数据库执行增删改查的所有操作
  - 数据表级别账号：对特定表执行增删改查等所有操作
  - 字段级别的权限：对某些表的特定字段进行操作
  - 存储程序级别的账号：对存储程序进行增删改查的操作
- 账户的操作主要包括创建账户、删除账户、修改密码、授权权限等

授予权限

需要使用实例级账户登录后操作，以root为例

主要操作包括：

- 查看所有用户
- 修改密码
- 删除用户

1. 查看所有用户

- 所有用户及权限信息存储在mysql数据库的user表中
- 查看user表的结构

```
desc user;
```

- 主要字段说明：
  - Host表示允许访问的主机
  - User表示用户名
  - authentication_string表示密码，为加密后的值

查看所有用户

```
select host,user,authentication_string from user;
```

2. 创建账户、授权

- 需要使用实例级账户登录后操作，以root为例
- 常用权限主要包括：create、alter、drop、insert、update、delete、select
- 如果分配所有权限，可以使用all privileges

2.1 创建账户&授权

```
grant 权限列表 on 数据库 to '用户名'@'访问主机' identified by '密码';
```

2.2 示例1

创建一个 `laowang` 的账号，密码为 `123456` ，只能通过本地访问, 并且只能对 `jd` 数据库中的所有表进行 `读` 操作

step1：使用root登录

```
mysql -uroot -p
回车后写密码，然后回车
```

step2：创建账户并授予所有权限

```
grant select on jd.* to 'laowang'@'localhost' identified by '123456';
```

说明

- 可以操作python数据库的所有表，方式为: `jd.*`
- 访问主机通常使用 百分号% 表示此账户可以使用任何ip的主机登录访问此数据库
- 访问主机可以设置成 localhost或具体的ip，表示只允许本机或特定主机访问

- 查看用户有哪些权限

```
show grants for laowang@localhost;
```

step3：退出root的登录

```
quit
```

step4：使用laowang账户登录

```
mysql -ulaowang -p
回车后写密码，然后回车
```

2.3 示例2

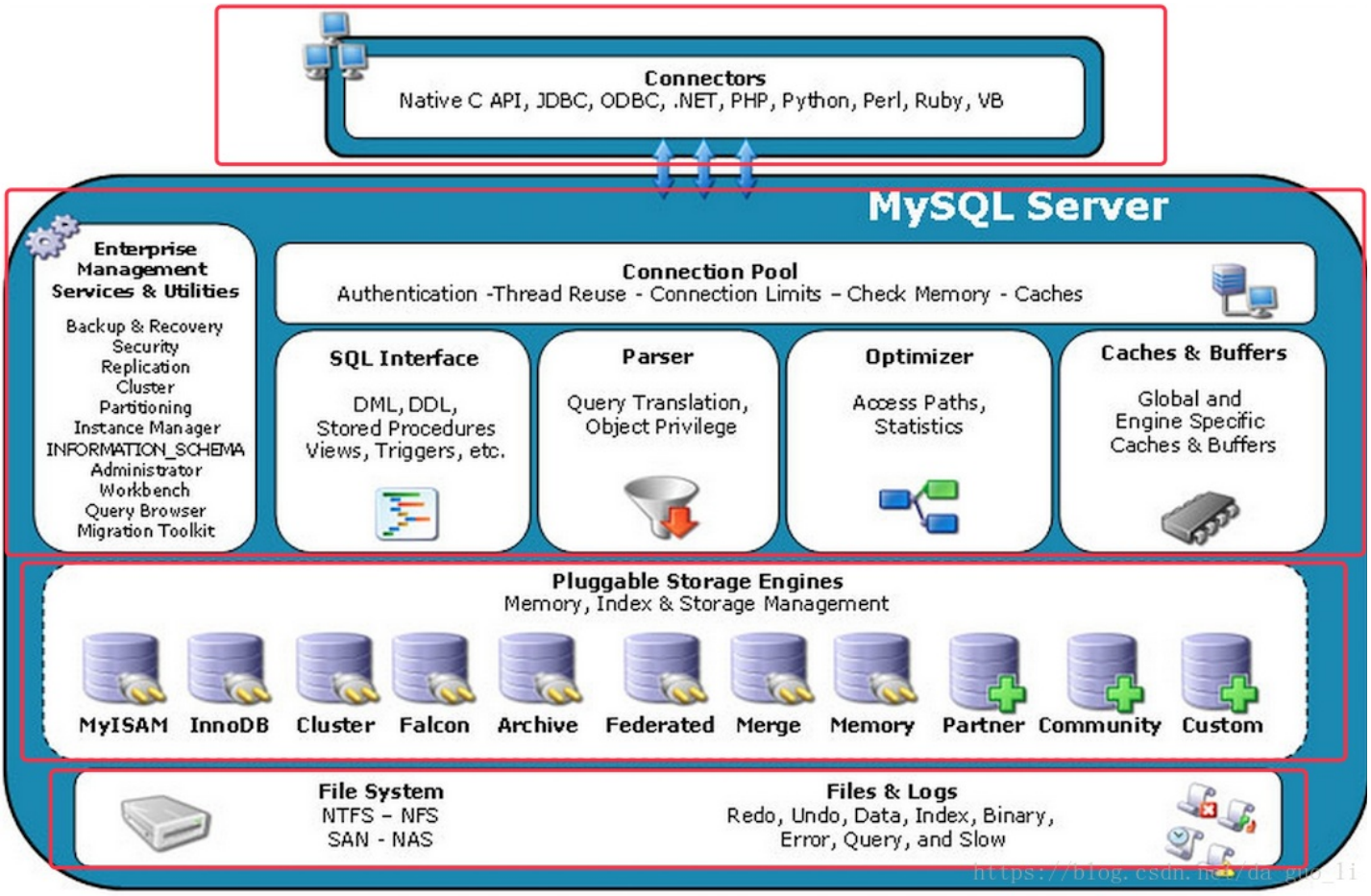
创建一个 `laoli` 的账号，密码为 `12345678` ，可以任意电脑进行链接访问，并且对 `jd` 数据库中的所有表拥有所有权限

```
grant all privileges on jd.* to "laoli"@"%" identified by "12345678"
```

6-数据库存储引擎

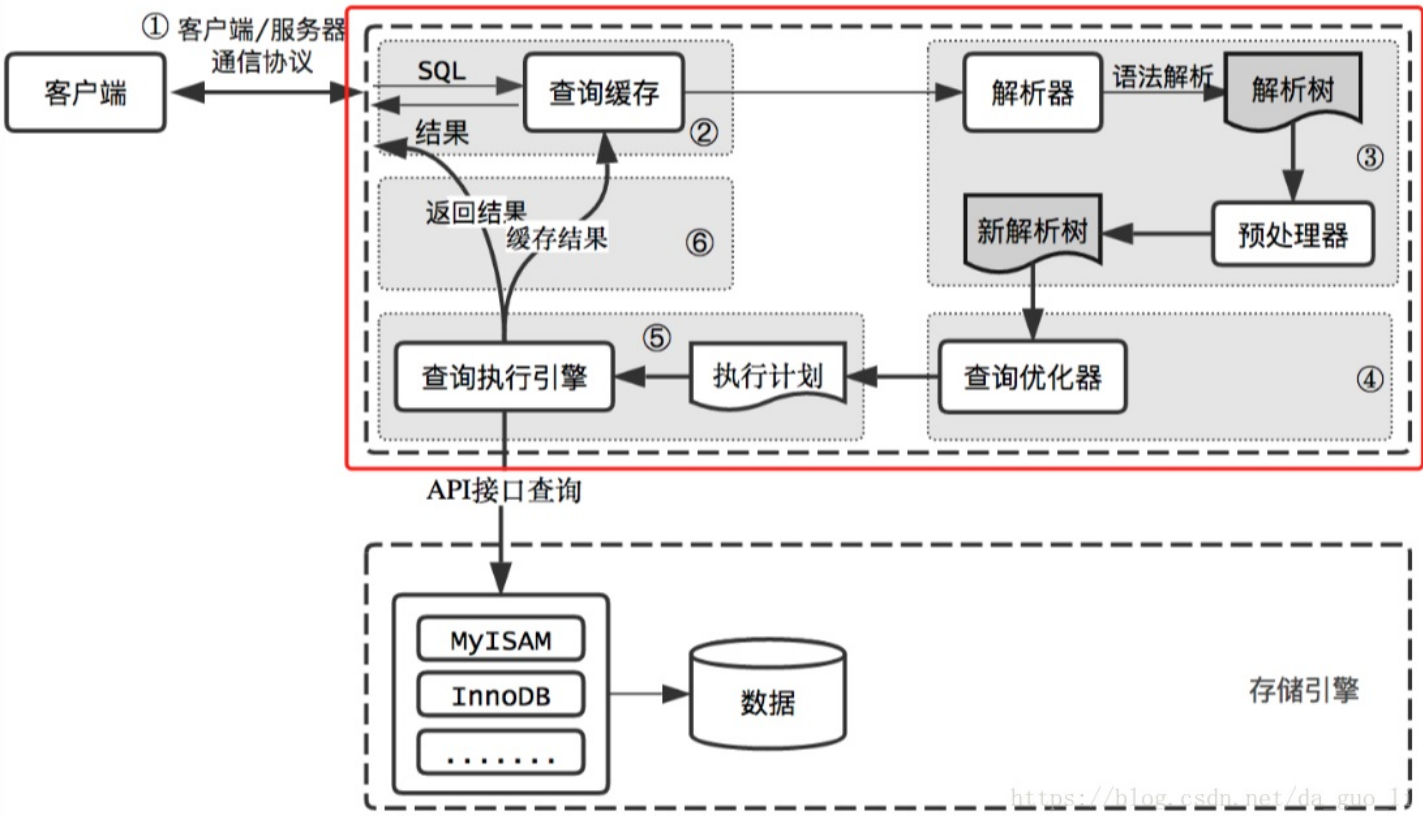
逻辑教育
Logic education

数据库存储引擎



服务层

第二层服务层是MySQL的核心，MySQL的核心服务层都在这一层，查询解析，SQL执行计划分析，SQL执行计划优化，查询缓存。以及跨存储引擎的功能都在这一层实现：存储过程，触发器，视图等。通过下图来观察服务层的内部结构：



存储引擎层

负责MySQL中数据的存储与提取。 服务器中的查询执行引擎通过API与存储引擎进行通信，通过接口屏蔽了不同存储引擎之间的差异。MySQL采用插件式的存储引擎。MySQL为我们提供了许多存储引擎，每种存储引擎有不同的特点。我们可以根据不同的业务特点，选择最适合的存储引擎。如果对于存储引擎的性能不满意，可以通过修改源码来得到自己想要达到的性能。例如阿里巴巴的X-Engine，为了满足企业的需求facebook与google都对InnoDB存储引擎进行了扩充。

查看存储引擎

```
show engines;
```

6-1-MyISAM存储引擎

MySQL引擎之MyISAM

MySQL5.5之前的版本默认存储引擎

MyISAM存储引擎表由MYD(数据文件)和MYI(索引文件)组成

什么是锁？

锁主要作用是管理共享资源的并发访问

锁用于实现事务的隔离性

锁的类型

> 共享锁(也称读锁),针对同一份数据,多个读操作可以同时进行而不会互相影响

> 独占锁(也称写锁),当前写操作没有完成前,它会阻断其他写锁和读锁

锁的粒度

> 表级锁

> 行级锁

MyISAM存储引擎特性

1.并发性与锁级别

2.表损坏修复

3.MyISAM表支持数据压缩

```
myisampack -b -f myIsam.MYI
```

MyISAM存储引擎限制

- 版本 < MySQL5.0时默认表大小为4G,如存储大表则要修改MAX_Rows和AVG_ROW_LENGTH
- 版本 > MySQL5.0时默认支持256TB

适合场景

> 1.非事务型应用

> 2.只读类应用

6-2-Innodb存储引擎

MySQL引擎之Innodb

MySQL5.5 及之后版本默认存储引擎,支持事务的ACID特性

Innodb使用表空间进行数据存储

```
innodb_file_per_table
```

ON:独立表空间,tablename.ibd

OFF:系统表空间:ibdataX X是一个数字

系统表空间和独立表空间如何选择?

- 系统表空间会产生IO瓶颈,刷新数据的时候是顺序进行的所以会产生文件的IO瓶颈
- 独立表空间可以同时向多个文件刷新数据

Innodb存储引擎的特性

1.支持事务的ACID特性

2.Innodb支持行级锁,可以最大程度的支持并发

MyISAM和InnoDB对比

对比项	MyISAM	InnoDB
主外键	不支持	支持
事务	不支持	支持
行表锁	表锁，即使操作一条记录也会锁住整个表，不适合高并发的操作	行锁,操作时只锁某一行，不对其它行有影响， 适合高并发的操作
缓存	只缓存索引，不缓存真实数据	不仅缓存索引还要缓存真实数据，对内存要求较高，而且内存大小对性能有决定性的影响
表空间	小	大
关注点	性能	事务
默认安装	Y	Y

6-3-CSV存储引擎

MySQL引擎之CSV

文件系统存储特点

数据以文本方式存储在文件中

.CSV文件存储表内容

.CSM文件存储表的元数据如表状态和数据量

.frm文件存储表结构信息

特点

以CSV格式进行数据存储

使用场景

适合做为数据交换的中间表

6-4-Memory存储引

MySQL引擎之Memory

也称HEAP存储引擎,所以数据保存在内存中,如果MySQL服务重启数据会丢失,但是表结构会保存下来

功能特点

- 支持HASH索引和BTree索引
- 所有字段都为固定长度 varchar(10)=char(10)
- 不支持BLOB和TEXT等大字段
- Memory存储引擎使用表级锁

如何选择存储引擎

大部分情况下，InnoDB都是正确的选择，可以简单地归纳为一句话“除非需要用到某些InnoDB不具备的特性，并且没有其他办法可以替代，否则都应该优先选择InnoDB引擎

参考条件

事务

> 如果应用需要事务支持，那么InnoDB(或者XtraDB)是目前最稳定并且经过验证的选择

备份

> 如果可以定期地关闭服务器来执行备份，那么备份的因素可以忽略。反之，如果需要在线热备份，那么选择InnoDB就是基本的要求

崩溃恢复

> MyISAM崩溃后发生损坏的概率比InnoDB要高很多，而且恢复速度也要慢

应用举例

- 日志型应用
- 只读或者大部分情况下只读的表
- 订单处理

7-MySQL基准测试

什么是基准测试

基准测试是一种测量和评估软件性能指标的活动用于建立某个时刻的性能基准,以便当系统发生软硬件变化时重新进行基准测试以评估变化对性能的影响

基准测试是针对系统设置的一种压力测试

基准测试特点

直接、简单、易于比较,用于评估服务器的处理能力

可能不关心业务逻辑,所使用的查询和业务的真实性可以和业务环境没关系

压力测试特点

- 对真实的业务数据进行测试,获得真实系统所能承受的压力
- 需要针对不同主题,所使用的数据和查询也是真实用到的
- 基准测试是简化了的压力测试

基准测试的目的

- 建立MySQL服务器的性能基准线,确定当前MySQL服务器运行情况,确定优化之后的效果
- 模拟比当前系统更高的负载,已找出系统的扩展瓶颈,可以增加数据库并发,观察QPS(每秒处理的查询数),TPS(每秒处理的事务数)变化,确定并发量与性能最优的关系
- 测试不同的硬件、软件和操作系统配置
- 证明新的硬件设备是否配置正确

如何进行基准测试

对整个系统进行基准测试

优点

- 能够测试整个系统的性能,包括web服务器缓存、数据库等
- MySQL并不总是出现性能问题的瓶颈,如果只关注MySQL可能忽略其他问题,能反映出系统中各个组件接口间的性能问题体现真实性能状况

缺点

基准测试最重要的就是简单,可能对不同的方案进行测试,找到最优的方案,基准测试进行的时间一定要短,否则就要花费大量的时间进行基准测试

- 测试设计复杂,消耗时间长

对MySQL进行基准测试

优点

- 测试设计简单,所耗费时间短

缺点

- 无法全面了解整个系统的性能基线

MySQL基准测试的常见指标

- 单位时间内处理的事务数(TPS)
- 单位时间内处理的查询数(QPS)

MySQL基准测试工具

MySQL基准测试之mysqlslap

- 可以模拟服务器负载,并输出相关统计信息

常用参数说明

- --auto-generate-sql 由系统自动生成SQL脚本进行测试
- --auto-generate-sql-add-autoincrement 在生成的表中增加自增ID
- --auto-generate-sql-load-type 指定测试中使用的查询类型 读写或者混合,默认是混合
- --auto-generate-sql-write-number 指定初始化数据时生成的数据量
- --concurrency 指定并发线程的数量 1,10,50,200
- --engine 指定要测试表的存储引擎,可以用逗号分割多个存储引擎
- --no-drop 指定不清理测试数据
- --iterations 指定测试运行的次数 指定了这个不能指定no-drop
- --number-of-queries 指定每一个线程执行的查询数量
- --debug-info 指定输出额外的内存及CPU统计信息

- --number-int-cols 指定测试表中包含的INT类型列的数量
- --number-char-cols 指定测试表中包含的varchar类型的数量
- --create-schema 指定了用于执行测试的数据库的名字
- --query 用于指定自定义SQL的脚本
- --only-print 并不运行测试脚本,而是把生成的脚本打印出来

```
mysqlslap --concurrency=1,50,100,200 --iterations=3 --number-int-cols=5 --number
-char-cols=5 --auto-generate-sql --auto-generate-sql-add-autoincrement --engine=
myisam,innodb --number-of-queries=10 --create-schema=test
```

sysbench测试工具：https://www.cnblogs.com/kismetv/archive/2017/09/30/7615738.html

	MyISAM	InnoDB
存储结构	每张表被存放在三个文件： 1. frm-表格定义 2. MYD(MYData)-数据文件 3. MYI(MYIndex)-索引文件	所有的表都保存在同一个数据文件中（也可能是多个文件，或者是独立的表空间文件），InnoDB表的大小只受限于操作系统文件的大小，一般为2GB
存储空间	MyISAM可被压缩，存储空间较小	InnoDB的表需要更多的内存和存储，它会在主内存中建立其专用的缓冲池用于高速缓冲数据和索引
可移植性、备份及恢复	由于MyISAM的数据是以文件的形式存储，所以在跨平台的数据转移中会很方便。在备份和恢复时可单独针对某个表进行操作	免费的方案可以是拷贝数据文件、备份 binlog，或者用 mysqldump，在数据量达到几十G的时候就相对痛苦了
事务安全	不支持 每次查询具有原子性	支持 具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表
AUTO_INCREMENT	MyISAM表可以和其他字段一起建立联合索引	InnoDB中必须包含只有该字段的索引
SELECT	MyISAM更优	
INSERT		InnoDB更优

UPDATE		InnoDB更优
DELETE		InnoDB更优 它不会重新建立表，而是一行一行的删除
COUNT without WHERE	MyISAM更优。因为MyISAM保存了表的具体行数	InnoDB没有保存表的具体行数，需要逐行扫描统计，就很慢了
COUNT with WHERE	一样	一样，InnoDB也会锁表
锁	只支持表锁	支持表锁、行锁 行锁大幅度提高了多用户并发操作的新能。但是InnoDB的行锁，只是在WHERE的主键是有效的，非主键的WHERE都会锁全表的
外键	不支持	支持
FULLTEXT全文索引	支持	不支持 可以通过使用Sphinx从InnoDB中获得全文索引，会慢一点

8-explain分析SQL语句

8-1-影响服务器性能的几个方面

影响服务器性能的几个方面

1.服务器硬件

2.服务器的操作系统

3.数据库存储引擎的选择

4.数据库参数配置

5.数据库结构设计和SQL语句

SQL性能下降原因

- 查询语句写的不好
- 索引失效
- 关联查询太多join
- 服务器调优及各个参数设置

SQL加载顺序

手写SQL的顺序

```
select distinct
    <select _list>
from
    <left_table>
join  <right_table> on <join_codition>
where
    <where_condition>
group by
    <group_by_list>
having
    <having_condition>
order by
    <order_by_condition>
```

```
limit <limit number>
```

机读的SQL顺序

```
1 FROM <left_table>
2 ON <join_condition>
3 <join_type> JOIN <right_table>
4 WHERE <where_condition>
5 GROUP BY <group_by_list>
6 HAVING <having_condition>
7 SELECT
8 DISTINCT <select_list>
9 ORDER BY <order_by_condition>
10 LIMIT <limit_number>
```

MySQL常见瓶颈

- CPU:CPU在饱和的时候一般发生在数据装入内存或从磁盘读取数据的时候
- IO:磁盘I/O瓶颈发生在装入数据远大于内存容量的时候
- 服务器硬件的性能瓶颈

8-2-explain分析SQL

explain

explain是什么？

使用explain关键字可以模拟优化器执行SQL查询语句,从而知道MySQL是如何处理你的SQL语句的。

explain能干嘛？

- 表的读取顺序
- 数据读取操作的操作类型
- 那些索引可以使用
- 那些索引被实际使用
- 表之间的引用
- 每张表有多少行被优化器查询

explain怎么玩？

```
explain + SQL语句
```

explain字段解释

id表的读取顺序

select查询的顺序号,包含一组数字,表示查询中执行select子句或操作表的顺序

两种情况:

1.id相同,执行顺序由上至下

2.id不同,如果是子查询,id的序号会递增,id值越大优先级越高,越先被执行

select_type---数据读取操作的操作类型

1.simple 简单的select查询,查询中不包含子查询或者union

2.primary 查询中若包含任何复杂的子部分,最外层查询则被标记。就是最后加载的那个

3.subquery 在select或where列表中包含了子查询

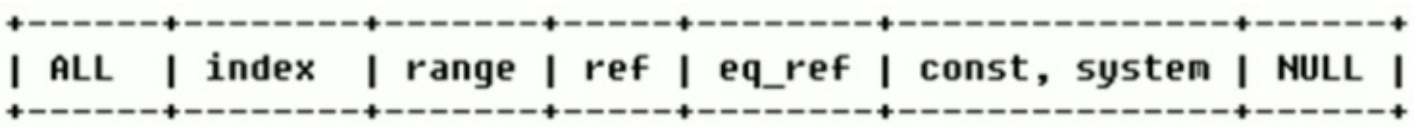
4.union 若第二个select出现在union之后,则被标记为union

5.union result 从union表获取结果的select

table---显示这一行的数据时关于那张表的

partitions-查询访问的分区

type



从最好到最差依次是

```
system > const > eq_ref > ref > range > index > ALL
```

- system,表只有一行记录(等于系统表),这是const类型的特例
- const表示通过索引一次就找到了,const用于比较primary key
- eq_ref,唯一索引扫描,对于每个索引键,表中只有一条记录与之匹配
- ref,非唯一性索引扫描,返回匹配某个单独值得所有行,本质上也是一种索引访问,它返回所有匹配某个单独值的行
- range,只检索给定范围的行,使用一个索引来选择行。
- index,Full Index Scan,index与ALL区别为index类型只遍历索引树。
- ALL,将遍历全表找到匹配的行

possible_keys

显示可能应用在这张表中的索引,一个或多个。

key

实际使用的索引。如果为null,则没有使用索引

查询中若使用了覆盖索引,则该索引仅出现在key列表中

key_len

表示索引中使用的字节数,可通过该列计算查询中使用的索引长度。在不损失精确性的情况下,长度越短越好

key_len显示的值为索引字段的最大可能长度,并非实际使用长度,即key_len是根据表定义计算而得,不是通过表内检索出的

索引长度计算

```
varchr(24)变长字段且允许NULL
24*(Character Set：utf8=3,gbk=2,latin1=1)+1(NULL)+2(变长字段)

varchr(10)变长字段且不允许NULL
10*(Character Set：utf8=3,gbk=2,latin1=1)+2(变长字段)

char(10)固定字段且允许NULL
10*(Character Set：utf8=3,gbk=2,latin1=1)+1(NULL)

char(10)固定字段且不允许NULL
10*(Character Set：utf8=3,gbk=2,latin1=1)
```

ref

显示索引那一列被使用到了

rows

根据表统计信息及索引选用情况,大致估算出找到所需的记录所需要读取的行数

```
mysql> explain select * from t1, t2 where t1.id = t2.id and t2.col1 = 'ac';
+----+-------------+-------+--------+---------------+---------+---------+--------------+------+-------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref          | rows | Extra       |
+----+-------------+-------+--------+---------------+---------+---------+--------------+------+-------------+
|  1 | SIMPLE      | t2    | ALL    | PRIMARY       | NULL    | NULL    | NULL         |  640 | Using where |
|  1 | SIMPLE      | t1    | eq_ref | PRIMARY       | PRIMARY | 4       | shared.t2.ID |    1 |             |
+----+-------------+-------+--------+---------------+---------+---------+--------------+------+-------------+
2 rows in set (0.00 sec)

mysql> create index idx_col1_col2 on t2(col1,col2);
Query OK, 1001 rows affected (0.17 sec)
Records: 1001  Duplicates: 0  Warnings: 0

mysql> explain select * from t1, t2 where t1.id = t2.id and t2.col1 = 'ac';
+----+-------------+-------+------+-----------------------+---------------+---------+-------+------+
| id | select_type | table | type | possible_keys         | key           | key_len | ref   | rows |
+----+-------------+-------+------+-----------------------+---------------+---------+-------+------+
|  1 | SIMPLE      | t2    | ref  | PRIMARY,idx_col1_col2 | idx_col1_col2 | 195     | const |  142 |
```

Extra

包含不适合在其他列中显示但十分重要的额外信息

- Using filesort,说明mysql会对数据使用一个外部的索引排序,而不是按照表内的索引顺序进行读取,MySQL中无法利用索引完成的排序操作称为"文件排序"
- Using temporary,使用临时表保存中间结果,MySQL在对查询结果排序时使用临时表。
- Using index,使用了索引,避免了全表扫描
- Using where,使用了where过滤

- Using join buffer,使用了连接缓存
- impossible where,不可能的条件,where子句的值总是false

9-索引优化案例

单表优化

建表

```
create table article(
    id int unsigned not null primary key auto_increment,
    author_id int unsigned not null,
    category_id int unsigned not null,
    views int unsigned not null,
    comments int unsigned not null,
    title varchar(255) not null,
    content text not null
);
```

插入数据

```
insert into article(`author_id`,`category_id`,`views`,`comments`,`title`,`conte
nt`) values
(1,1,1,1,'1','1'),
(2,2,2,2,'2','2'),
(1,1,3,3,'3','3');
```

查询category_id为1且comments大于1的情况下,views最多的article_id

双表优化

建表

```
商品类别
create table class(
    id int unsigned not null primary key auto_increment,
    card int unsigned not null
```

```
);

图书表
create table book(
    bookid int unsigned not null auto_increment primary key,
    card int unsigned not null
);
```

驱动表的概念，mysql中指定了连接条件时，满足查询条件的记录行数少的表为驱动表；如未指定查询条件，则扫描行数少的为驱动表。mysql优化器就是这么粗暴以小表驱动大表的方式来决定执行顺序的。

10-索引优化

键表SQL

```
create table staffs(
    id int primary key auto_increment,
    name varchar(24) not null default "",
    age int not null default 0,
    pos varchar(20) not null default "",
    add_time timestamp not null default CURRENT_TIMESTAMP
)charset utf8;
```

```
create table user(
    id int not null auto_increment primary key,
    name varchar(20) default null,
    age int default null,
    email varchar(20) default null
) engine=innodb default charset=utf8;
```

插入数据

```
insert into staffs(`name`,`age`,`pos`,`add_time`) values('z3',22,'manager',now()
);
insert into staffs(`name`,`age`,`pos`,`add_time`) values('July',23,'dev',now());

insert into staffs(`name`,`age`,`pos`,`add_time`) values('2000',23,'dev',now());
```

```
insert into user(name,age,email) values('1aa1',21,'b@163.com');
insert into user(name,age,email) values('2aa2',22,'a@163.com');
insert into user(name,age,email) values('3aa3',23,'c@163.com');
```

```
insert into user(name,age,email) values('4aa4',25,'d@163.com');
```

建立复合索引

```
create index idx_staffs_nameAgePos on staffs(name,age,pos);
```

口诀

全值匹配我最爱,最左前缀要遵守

带头大哥不能死,中间兄弟不能断

索引列上少计算,范围之后全失效

like百分写最右,覆盖索引不写星

不等空值还有or,索引失效要少用

varchar引号不可丢,SQL高级也不难

练习

假设index(a,b,c)

where语句	索引是否被使用到
where a = 3	Y,使用到a
where a = 3 and b = 5	Y,使用到a,b
where a = 3 and b = 5 and c = 4	Y,使用到a,b,c
where b = 3 或 where b = 3 and c = 4 或 where c = 4	N
where a = 3 and c = 5	使用到a,c没有被使用,b中间断了
where a = 3 and b > 4 and c = 5	使用到了a,b
where a = 3 and b like 'kk%' and c = 4	使用到了a,b,c
where a = 3 and b like '%kk' and c = 4	使用到了a
where a = 3 and b like '%kk%' and c = 4	使用到了a

11-排序优化

排序优化

分析

- 1.观察,至少跑一天,看看生产的慢SQL情况
- 2.开启慢查询日志,设置阈值,比如超过5秒钟的就是慢SQL,并抓取出来
- 3.explain + 慢SQL分析
- 4.show profile
- 5.进行SQL数据库服务器的参数调优(运维orDBA来做)

总结

- 1.慢查询的开启并捕获
- 2.explain+慢SQL分析
- 3.show profile查询SQL在MySQL服务器里面的执行细节
- 4.SQL数据库服务器的参数调优

永远小表驱动大表

```
for i in range(5):
    for j in range(1000):
        pass

for i in range(1000):
    for j in range(5):
        pass
```

order by优化

```
order by子句,尽量使用index方式排序,避免使用filesort方式排序
```

1.建表，插入测试数据

```
create table tbla(
    age int,
    birth timestamp not null
);

insert into tbla(age,birth) values(22,now());
insert into tbla(age,birth) values(23,now());
insert into tbla(age,birth) values(24,now());
```

2.建立索引

```
create index idx_tbla_agebrith on tbla(age,birth);
```

3.分析

MySQL支持两种方式的排序,filesort和index,index效率高,MySQL扫描索引本身完成排序。filesort方式效率较低

order by 满足两种情况下,会使用index方式排序

- 1.order by 语句使用索引最左前列
- 2.使用where子句与order by子句条件组合满足索引最左前列

filesort有两种算法-双路排序和单路排序

双路排序,MySQL4.1之前是使用双路排序,字面意思就是两次扫描磁盘,最终得到数据,读取行指针和order by列,对他们进行排序,然后扫描已经排序好的列表,按照列表中的值重新从列表中读取对应的数据输出

单路排序,从磁盘读取查询需要的所有列,按照order by列在buffer对他们进行排序,然后扫描排序后的列表进行输出,它的效率更快一些,避免了第二次读取数据,并且把随机IO变成了顺序IO,但是它会使用更多的空间

优化策略调整MySQL参数

```
增加sort_buffer_size参数设置
增大max_lenght_for_sort_data参数的设置
```

提高order by的速度

- order by时select * 是一个大忌,只写需要的字段
    - 当查询的字段大小总和小于max_length_for_sort_data而且排序字段不是text|blob类型时,会用改进后的算法--单路排序

    - 两种算法的数据都有可能超出sort_buffer的容量,超出之后,会创建tmp文件进行合并排序,导致多次I/O
- 尝试提高sort_buffer_size
- 尝试提高max_length_for_sort_data

练习

```
索引 a_b_c(a,b,c)

order by a,b
order by a,b,c
order by a desc,b desc,c desc

where a = const order by b,c
where a = const and b = const order by c
where a = const and b > const order by b,c

order by a asc,b desc,c desc
where g = const order by b,c
where a = const order by c
where a = const order by by a,d
```

12-慢查询日志

慢查询日志

- MySQL的慢查询日志MySQL提供的一种日志记录,它用来记录在MySQL中响应时间超过阈值的语句,具体指运行时间超过 `long_query_time` 值得SQL,则会被记录到慢查询日志中
- 具体指运行时间超过 `long_query_time` 值得SQL,则会被记录到慢查询日志中。 `long_query_time` 的默认值为10,意思是运行10秒以上的语句。
- 由他来查看那些SQL超出了我们的最大忍耐时间值,比如一条SQL执行超过5秒钟,我们就算是慢SQL,希望能收集超过5秒的SQL,结合之前explain进行全面分析

怎么玩

- 默认情况下,MySQL数据库没有开启慢查询日志,需要我们手动来设置这个参数
- 当然如果不是调优需要的话,一般不建议启动该参数,因为开启慢查询日志会或多或少带来一定的性能影响。慢查询日志支持将日志记录写入文件

慢查询日志工具

s:表示按照何种方式排序

c:访问次数

l:锁定时间

r:返回记录

t:查询时间

al:平均锁定时间

ar:平均返回记录数

t:即为返回前面多少条的数据

得到返回记录集最多的10个SQL

mysqldumpslow -s r -t 10 D:/phpStudy/PHPTutorial/MySQL/slow_log.txt

得到访问次数最多的10个SQL

mysqldumpslow -s c -t 10 D:/phpStudy/PHPTutorial/MySQL/slow_log.txt

批量插入数据

函数和存储过程

部门表

```
create table dept(
            id int primary key auto_increment,
            deptno mediumint not null,
            dname varchar(20) not null,
            loc varchar(13) not null
)engine=innodb default charset=gbk;
```

员工表

```
create table emp(
            id int primary key auto_increment,
            empno mediumint not null,
            ename varchar(20) not null,
            job varchar(9) not null,
            mgr mediumint not null,
            hiredate DATE not null,
            sal decimal(7,2) not null,
            comm decimal(7,2) not  null,
            deptno mediumint not null
)engine=innodb default charset=gbk;
```

创建函数

创建函数，假如报错：this function has none of DETERMINISTIC... 查看参数

```
set global log_bin_trust_function_creators=1;
```

随机产生字符串

```
DELIMITER $$
CREATE FUNCTION rand_string(n INT) RETURNS VARCHAR(255)
BEGIN
DECLARE chars_str VARCHAR(100) DEFAULT 'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLM
NOPQRSTUVWXYZ';
DECLARE return_str VARCHAR(255) DEFAULT '';
```

```
DECLARE i INT DEFAULT 0;
WHILE i < n DO
SET return_str = CONCAT(return_str,SUBSTRING(chars_str,FLOOR(1+RAND()*52),1));
SET i = i + 1;
END WHILE;
RETURN return_str;
END $$
```

随机产生部门编号

```
DELIMITER $$
CREATE FUNCTION rand_num() RETURNS INT(5)
BEGIN
DECLARE i INT DEFAULT 0;
SET i = FLOOR(100+RAND()*10);
RETURN i;
END $$
```

创建存储过程

插入数据

```
DELIMITER $$
CREATE PROCEDURE insert_emp(IN START INT(10),IN max_num INT(10))
BEGIN
DECLARE i INT DEFAULT 0;
SET autocommit = 0;
REPEAT
SET i = i + 1;
INSERT INTO emp(empno,ename,job,mgr,hiredate,sal,comm,deptno) VALUES ((START+i),
rand_string(6),'SALESMAN',0001,CURDATE(),2000,400,rand_num());
UNTIL i = max_num
END REPEAT;
COMMIT;
END $$
```

插入数据

```
DELIMITER $$
CREATE PROCEDURE insert_dept(IN START INT(10),IN max_num INT(10))
BEGIN
DECLARE i INT DEFAULT 0;
SET autocommit = 0;
REPEAT
SET i = i + 1;
INSERT INTO dept(deptno,dname,loc) VALUES ((START + i),rand_string(10),rand_str
```

```
ing(8));
UNTIL i = max_num
END REPEAT;
COMMIT;
END $$
```

调用存储过程

```
delimiter ;
call insert_dept(100,10);

delimiter ;
call insert_emp(100001,500000);
```

13-Show Profile进行SQL分析

Show Profile进行SQL分析

是MySQL提供可以用来分析当前会话中语句执行的资源消耗情况,可以用于SQL的调优的测量

默认情况下,参数处于关闭状态,并保持最近15次的运行结果。

Show Profile分析步骤

- 1.是否支持,看看当前MySQL版本是否支持
- 2.开启功能,默认是关闭,使用前需要开启

type

```
all          显示所有的开销信息
block io     显示块IO相关开销
cpu          显示CPU相关开销信息
ipc          显示发送和接收相关开销信息
memory       显示内存相关开销信息
page faults 显示页面错误相关开销信息
```

参数注意

```
converting HEAP to MyISAM查询结果太大,内存都不够用了往磁盘上搬

Creating tmp table 创建临时表

Copying to tmp table on disk 把内存中临时表复制到磁盘,危险

locked
```

全局查询日志

开启命令

```
set global general_log = 1;
```

将SQL语句写到表中

```
set global log_output = 'TABLE';
```

你所编写的SQL语句,会记录到MySQL库里的genral_log表

```
select * from mysql.general_log;
```

14-数据库锁

数据库锁

- 表锁
- 行锁
- 间隙锁

锁是计算机协调多个进程或线程并发访问某一资源的机制

表锁(偏读)

偏向MyISAM存储引擎,开销小,加锁快;无死锁,锁定粒度大,发送锁冲突的概率最高,并发度低

表锁案例

```
create table mylock(
    id int not null primary key auto_increment,
    name varchar(20)
)engine myisam;

insert into mylock(name) values('a');
insert into mylock(name) values('b');
insert into mylock(name) values('c');
insert into mylock(name) values('d');
insert into mylock(name) values('e');
```

手动增加表锁

```
lock table 表名字 read(write),表名字2 read(write);
```

释放表锁

```
unlock tables;
```

表锁总结

MyISAM在执行查询语句(select)前,会自动给涉及的所有表加读锁,在执行增删改操作前,会自动给涉及的表加写锁

- 对MyISAM表的读操作(加读锁),不会阻塞其他进程对同一表的读请求,但会阻塞对同一表的写请求。只有当读锁释放后,才会执行其他进行的写操作
- 对MyISAM表的写操作(加写锁),会阻塞其他进程对同一表的读和写操作,只有当写锁释放后,才会执行其他进程的读写操作

行锁(偏写)

偏向InnoDB存储引擎,开销大,加锁慢,会出现死锁。锁定粒度最小,发生锁冲突的概率最低,并发度也最高

InnoDB与MyISAM的最大不同点,支持事务,采用了行级锁

行锁案例

```
create table test_innodb_lock(a int(11),b varchar(16))engine=innodb;

insert into test_innodb_lock values(1,'b2');
insert into test_innodb_lock values(3,'3');
insert into test_innodb_lock values(4,'4000');
insert into test_innodb_lock values(5,'5000');

create index idx_test_innodb_a on test_innodb_lock(a);

create index idx_test_innodb_b on test_innodb_lock(b);
```

如何分析行锁定

```
通过检查innodb_row_lock状态变量来分析系统上的行锁争夺情况

show status like 'innodb_row_lock%';
```

各个状态量的说明

```
Innodb_row_lock_current_waits        当前正在等待锁定的数量
* Innodb_row_lock_time             从系统启动到现在锁定的总时间长度
* Innodb_row_lock_time_avg         每次等待所花费平均时间
Innodb_row_lock_time_max          从系统启动到现在等待最长的一次所花费的时间
* Innodb_row_lock_waits            系统启动后到现在总共等待的次数
```

什么是间隙锁

当我们用范围条件而不是相等条件检索数据,并请求共享或排他锁时,innodb会给符合条件的已有数据记录的索引项加锁, 对于键值在条件范围内但并不存在的记录,叫做"间隙"

innodb也会对这个"间隙"加锁,这种锁机制就是所谓的间隙锁

间隙锁的危害

因为SQL执行过程中通过范围查找的话,他会锁定整个范围内所有的索引值,即使这个键值并不存在

间隙锁有一个比较致命的弱点,就是当锁定以为范围键值之后,即使某些不存在的键值也会被无辜的锁定,而造成在锁定的时候无法插入锁定键值范围内的任何数据。在某些场景下这可能会对性能造成很大的危害

如何锁定一行

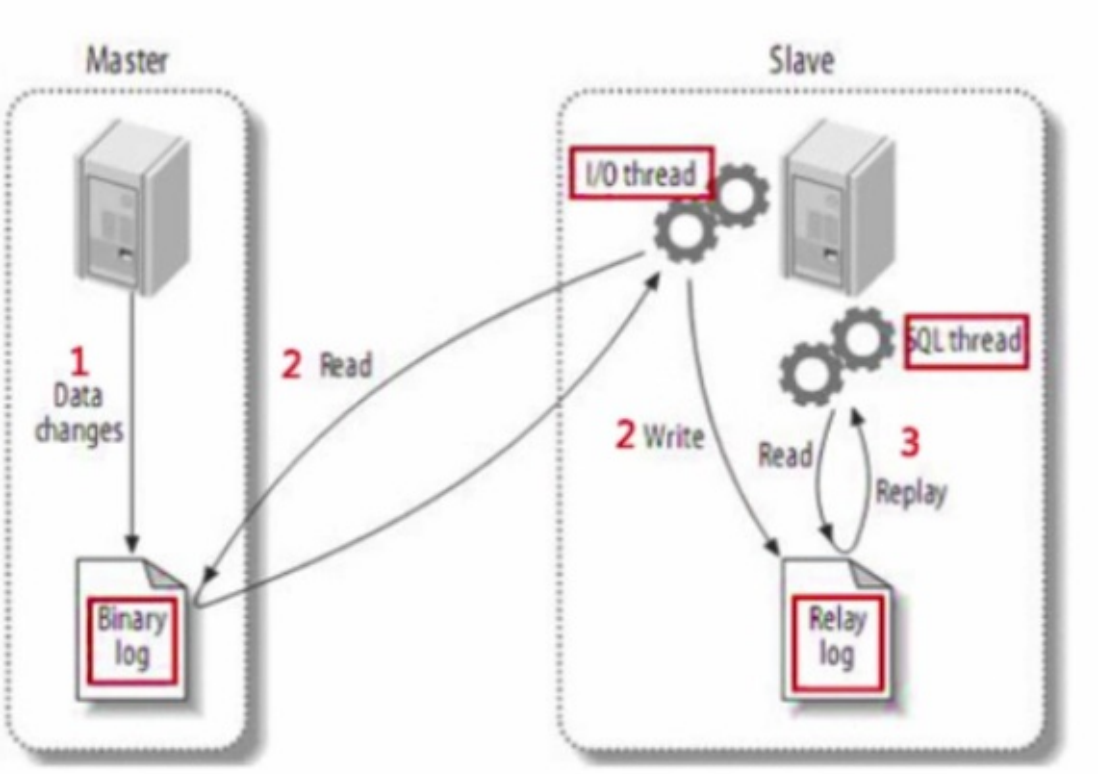
```
select * from test_innodb_lock where a = 8 for update;
```

15-主从复制

逻辑教育
Logic education

主从复制

复制的基本原理



复制的三步骤

- 1.master将改变记录到二进制日志。这些记录过程叫做二进制日志事件,binary log events
- 2.slave将master的binary log events拷贝到它的中继日志
- 3.slave重做中继日志中的事件,将改变应用到自己的数据库中。MySQL复制是异步的且串行的

复制的基本原则

- 1.每个slave只有一个master

- 2.每个slave只能有一个唯一的服务器ID
- 3.每个master可以有多个salve

一主一从常见配置

- MySQL版本一致且后台服务可以运行
- 主从主机可以相互通信
- 主从配置都在[mysqld]结点下,都是小写

主机配置文件-my.ini

```
server-id = 1        # [必须]主服务器唯一ID
log-bin = 自己本地的路径/mysqlbin    # [必须]启用二进制日志
log-err = 自己本地的路径/mysqlerr    # [可选] 启用错误日志

从机配置文件-mysqld.cnf      /etc/mysql/mysql.conf.d/mysqld.cnf
server-id = 1        # [必须]主服务器唯一ID
log-bin = 自己本地的路径/mysqlbin    # [可选]启用二进制日志
```

修改过配置文件之后,要重启MySQL服务

```
service mysql restart
```

主从都关闭防火墙

```
service iptables stop
```

在Windows主机上建立账户并授权slave

```
grant replication slave on *.* to 'zhangsan'@'从机数据库IP' identified by '12345
6';


show master status;

记录下File和position的值
```

配置Linux从机

```
change master to master_host = '192.168.0.161',
master_user = 'juran',
master_password = '123',
master_log_file = 'binlog.000004',
```

```
master_log_pos= 908;
```

测试是否配置成功

```
start slave;                    启动从服务器复制功能

show slave status\G

下面两个参数都是yes,则说明主从配置成功

slave_io_running:yes

slave_sql_running:yes
```

16-MySQL分区表

MySQL分区表

分区表的特点

在逻辑上为一个表,在物理上存储在多个文件中

```
create table `login_log`(
    login_id int(10) unsigned not null comment '登录用户id',
    login_time timestamp not null default current_timestamp,
    login_ip int(10) unsigned not null comment '登录类型'
)engine=innodb default charset=utf8 partition by hash(login_id) partitions 4;
```

分区键

分区引入了分区键的概念,分区键用于根据某个区间值、特定值、或者HASH函数值执行数据的聚集,让数据根据规则分布在不同的分区中。

分区类型

- RANGE分区
- LIST分区
- HASH分区

无论那种分区类型,要么分区表上没有主键/唯一键,要么分区表的主键/唯一键都必须包括分区键,也就是说不能使用主键/唯一字段之外的其他字段分区

RANGE分区

RANGE分区特点

- 根据分区键值的范围把数据行存储到表的不同分区中
- 多个分区的范围要连续,但是不能重叠

- 分区不包括上限,取不到上限值

建立RANGE分区

```
create table `login_log_range`(
    login_id int(10) unsigned not null comment '登录用户ID',
    login_time timestamp not null default CURRENT_TIMESTAMP,
    login_ip int(10) unsigned not null comment '登录ip'
)engine=innodb
partition by range(login_id)(
partition p0 values less than(10000),   # 实际范围0-9999
partition p1 values less than(20000),   # 实际范围10000-19999
partition p2 values less than(30000),
partition p3 values less than maxvalue  # 存储大于30000的数据
);
```

RANGE分区使用场景

- 分区键为日期或是时间类型
- 经常运行包含分区键的查询,MySQL可以很快的确定只有某一个或某些分区需要扫描,例如检索商品login_id小于10000的记录数,MySQL只需要扫描p0分区即可
- 定期按分区范围清理历史数据

HASH分区

HASH分区的特点

- 根据MOD(分区键,分区值)的值把数据行存储到表的不同分区内
- 数据可以平均的分布在各个分区中
- HASH分区的键值必须是一个INT类型的值,或是通过函数可以转为INT类型

如何建立HASH分区表

```
create table `login_log`(
    login_id int(10) unsigned not null comment '登录用户ID',
    login_time timestamp not null default CURRENT_TIMESTAMP,
    login_ip int(10) unsigned not null comment '登录ip'
)engine=innodb default charset=utf8 partition by hash(login_id) partitions 4;

create table `login_log`(
    login_id int(10) unsigned not null comment '登录用户ID',
    login_time timestamp not null default CURRENT_TIMESTAMP,
    login_ip int(10) unsigned not null comment '登录ip'
)engine=innodb default charset=utf8 partition by hash(UNIX_TIMESTAMP(login_time)
) partitions 4;
```

LIST分区

LIST分区特点

- 按分区键取值的列表进行分区
- 同范围分区一样,各分区的列表值不能重复
- 每一行数据必须能找到对应的分区列表,否则数据插入失败

建立LIST分区

```
create table `login_log_list`(
    login_id int(10) unsigned not null comment '登录用户ID',
    login_time timestamp not null default CURRENT_TIMESTAMP,
    login_ip int(10) unsigned not null comment '登录ip',
    login_type int(10) not null
)engine=innodb
partition by list(login_type)(
partition p0 values in(1,3,5,7,9),
partition p1 values in(2,4,6,8)
);
```

如何选择合适的分区方式

业务场景

1.用户每次登陆都会记录到日志表中

2.用户登录日志保存一年,一年后可以删除

```
create table `login_log_range`(
    login_id int(10) unsigned not null comment '登录用户id',
    login_time datetime  not null default current_timestamp,
    login_ip int(10) unsigned not null comment '登录ip'
)engine=innodb
partition by range(year(login_time))(
partition p0 values less than(2015),
partition p1 values less than(2016),
partition p2 values less than(2017)
);
```

插入数据

```
insert into login_log_range values
(1,'2015-01-25',1),
(2,'2015-07-25',2),
(3,'2015-06-25',3),
(4,'2016-03-25',2),
(5,'2016-02-25',1);
```

查询表

```
select table_name,partition_name,partition_description,table_rows from
information_schema.`partitions` where table_name = 'login_log_range'
```

修改分区-添加分区

```
alter table login_log_range add partition (partition p4 values less than(2018))
```

分区删除

```
alter table login_log_range drop partition p0;
```

使用分区表的注意事项

- 结合业务场景选择分区键,避免跨分区查询
- 对分区表进行查询最好在where从句中包含分区键
- 具有主键或唯一索引的表,主键或唯一索引必须是分区键的一部分