# 数据库

|  |
| --- |
| 分类：   * 关系型数据库 * 非关系型数据库 |

## Mysql安装

|  |
| --- |
| * 关闭防火墙：   systemctl stop firewalld  firewall-cmd --state ##查看状态   * 检查是否有自带的mysql   rpm -qa |grep mysql   * 下载mysql-5.6.45-linux-glibc2.12-x86\_64.tar.gz * 解压mysql 并重命名   tar -xvf MySQL-5.6.45-1.el7.x86\_64.rpm-bundle.tar   * 安装rpm包：   rpm -ivh MySQL\*.rpm  如出现：错误：依赖检测失败：  mariadb-libs 被 MySQL-shared-compat-5.6.45-1.el7.x86\_64 取代  就卸载掉：yum remove mysql-libs   * 安装完后初始化密码位置：      * 启动mysql   service mysql start   * 使用初始化密码登录并设置新密码：   mysql -uroot -p  set PASSWORD=PASSWORD('root');   * 退出并用新密码登录，授权远程登录权限   GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY 'root' with grant option;##蓝色root为密码  Flush privileges;  远程连接测试：mysql -h master2 -uroot -p  Mysql数据库默认存储路径：/var/lib/mysql    ps -ef|grep mysql 可以查看mysql的一些目录路径。    配置文件目录：/usr/share/mysql  Mysql启停脚本：/etc/init.d/mysql |

## SQL

|  |
| --- |
| SQL主要分为三种：  DDL:数据定义语言，create,drop,alter  DML:数据操作语言（insert,delete,update）内部有分了：DQL(select)  DCL:数据控制语言（grant/revoke/if else...） |

## DDL

|  |
| --- |
| 表，视图，索引  创建数据库：基本语法：create database + 数据库名称 + [库选项];  create database TBL\_ERROR\_CODE charset utf8;  **数据库的名称不能更改**  修改数据库的字符集：  alter database TBL\_ERROR\_CODE charset gbk;  查询库：  Show databases;  show databases like '%db%';###模糊查询  创建视图：  Create view temp\_view;  创建索引：  Create index test;  创建表：   1. create table [**if** not exists] + 表名( 2. 字段名称 数据类型, 3. …… 4. 字段名称 数据类型   /\* 最后后一行，不需要加逗号 \*/ 5. )[表选项];   如：  create table if not exists userinfo  (  id int primary key,  addr varchar(50),  age int  );  **【表选项】**：   * 字符集设定：charset/ character set+ 具体字符集，用来表示数据存储的编码格式，常用的字符集包括GBK和UTF8等。 * 校对集设定：collate+ 具体校对集，表示数据比较的规则，其依赖字符集。 * 存储引擎：engine+具体存储引擎，默认为InnoDB，常用的还有MyISAM.     **对表结构的修改：**  新增字段，基本语法：alter table + 表名 + add + [column] + 字段名 + 数据类型 + [列属性][位置];  修改字段，基本语法：alter table + 表名 + modify + 字段名 + 数据类型 + [列属性][位置];  重命名字段，基本语法：alter table + 表名 + change + 旧字段名 + 新字段名 + 数据类型 + [列属性][位置];  删除字段，基本语法：alter table + 表名 + drop+ 字段名; |

## DML

|  |
| --- |
|  |

## DQL

|  |
| --- |
|  |

## 中文数据问题

|  |
| --- |
| 客户端和服务端字符集不一致，可能导致插入的数据乱码无法识别。  现实服务端识别的字符集：  show character set;   1. -- 查看服务器默认的对外处理的字符集 2. show variables like 'character\_set%';   show variables like 'character\_set%';      永久设置编码：/etc/my.cnf,重启mysql服务 |

## 校对集

校对集必须提前设置，不然无效

|  |
| --- |
| 数据对比的方式：共有三种   * \_bin:binary，二进制比较，区分大小写； * \_cs:case sensitive，大小写敏感，区分大小写； * \_ci:case insensitive，大小写不敏感，不区分大小写。   全部校对集：  show collation;    如下：不同校对集区别，使用name 排序的不同 |

## 修改mysql配置文件

|  |
| --- |
| cp /usr/share/mysql/my-default.cnf /etc/my.cnf  配置字符集和校对集：只对修改后创建的库/表生效。  [mysqld]  character\_set\_server=utf8  character\_set\_client=utf8  collation-server=utf8\_general\_ci  [mysql]  default-character-set=utf8  [client]  default-character-set=utf8 |

## 数据类型

|  |
| --- |
| **整数型：**   * Tinyint:一个字节（常用） * Smallint:两个字节 * Int:4个字节（常用） * Bigint:8个字节   无符号定义：  alter table my\_int add int\_5 tinyint unsigned;  **小数型：**   * 浮点型（float 4个字节/double 8个字节）   直接使用float则表示：无小数部分，float(M,D),M表示总长度，D表示小数部分长度  alter table user add f1 float;  alter table user add f2 float(4,1);  insert into user(f1,f2) value(111.0,123.5555);     * **定点型（保证整数位不会四舍五入，能保证整数位的精度）**   非定点型：如定义float(5,2),插入：12.555,则插入结果为：**12.56**  **日期类型**   * Datetime:日期时间，格式：yyyy-MM-dd HH:mm:ss   如：’2017-05-06 13:15:00‘   * Date:日期，datetime的date部分   如：’2017-05-04‘   * Time:时间，时间段   如：’13:15:00‘   * Timestamp:时间戳,格式和datetime一样,当前行记录被更新，则时间戳类型字段更新为当前你时间。   如：’2017-05-06 13:15:00‘   * Year:年份有两种格式：year(2),year(4)   如：90,1990  **字符串型：**   * Char:定长，空间浪费，效率高（已经确定长度的字段）   如：char(4),最大长度255   * Varchar：变长，实际使用多少决定在于数据长度。效率低，节省空间   如：Varchar(10)   * Text：字符串比较长，超过255个字符，使用text.(存储文字) * Blob：也为存储较长字符串，格式：二级制（一般不用，存储路径即可） * Enum：枚举字符串，必须存储定义好的内容中的。   定义：enum('元素1','元素2','元素3'...)，例如enum('男','女','保密')；  枚举实际存储的是数值如：1,2,3而非字符串。节省空间，但效率不高，需要转换。   * Set：和枚举类似，实际存储数值。节省空间，但效率不高。   create table my\_set(  hobby set('音乐','电影','旅行','美食','摄影','运动','宠物')  )charset utf8;  如果插入：  insert into my\_set values ('电影,美食,宠物');  其实际存储的为：  选择中为1，否则为0：101001    将其倒置：二进制：100101= 74，存储的就是74. |

## Mysql数值长度

|  |
| --- |
| Mysql中记录不能超过65535个字节。这意味这varchar永远达不到理论最大值  各个字符集最大长度：  create table my\_utf8(  name varchar(21844)  )charset utf8;    create table my\_gbk(  name varchar(32766)  )charset gbk;  text文本字符串**不占用记录长度**，**额外存储**，但是text文本字符串也是属于记录的一部分，无论是在utf8还是在gbk字符集之中，其都占用记录中的**10个字节长度**，用来保存数据的地址以及长度。 |

## 列属性

|  |
| --- |
| 真正约束字段的是数据类型，但是比较单一，所以有列属性：  null、not null、default、primary key、unique key、auto\_increment和comment等    Comment：列描述  Default：默认值。如：  Create table test(age tinyint unsigned default 0)  使用默认值：insert into test values (default);  primary key：主键（不能重复的约束）  **单一主键设置：**  create table my\_pri1(  name varchar(20) not null comment '姓名',  number char(10) primary key comment '学号'  )charset utf8;  **符合主键：多个字段共同为主键**  create table my\_pri1(  name varchar(20) not null comment '姓名',  number char(10) comment '学号' ,  Primary key(name,number)  )charset utf8;  **表创建完后追加主键：需保证表中数据将要设置的主键字段是唯一的，不能重复。**  alter table my\_pri3 modify course char(10) primary key comment '课程编号'; -- 不建议使用  alter table my\_pri3 add primary key(course); -- **推荐使用**  **更改主键必须先删除后添加：**  alter table my\_pri3 drop primary key;  **Auto\_increment :自增长，每个表最多一个自增长字段，必须为数字型字段，字段必须是索引。一般和主键搭配使用。**  create table my\_auto(  id int primary key auto\_increment,  name varchar(20) not null  )charset utf8;  使用自增长：当字段给的是NULL或默认时就会触发自增长。  Insert into my\_auto value(null,’gp’);  Insert into my\_auto value(default,’gp11’);  查看下一次自增长的值：  show create table my\_auto;    修改自增长的下一个值：必须比当前自增长的最大值大才生效。  alter table my\_auto auto\_increment = 2;  自增长的初始值和步长变量：修改该值课改表初始值和步长  show variables like 'auto\_increment%';  +--------------------------+-------+  | Variable\_name | Value |  +--------------------------+-------+  | auto\_increment\_increment | 1 |  | auto\_increment\_offset | 1 |  +--------------------------+-------+  如下修改为会话级修改，针对库的，不是表的：  set auto\_increment\_increment = 5;  删除自增长：  alter table my\_auto modify id int ;  唯一性：主键一张表只能有一个字段，如果多个字段需要保证唯一性。唯一键与主键本质相同，区别在于：唯一键允许字段值为空，并且允许多个空值存在。  单一唯一字段：  create table my\_unique(  number char(10) unique comment '学号',  name varchar(20) not null  )charset utf8;  符合唯一字段：  create table my\_unique2(  number char(10) not null,  name varchar(20) not null,  unique key(number)  )charset utf8;  在创建表之后：  alter table my\_unique3 add unique key(number);  删除唯一约束：  alter table my\_unique3 drop index number; |

## 基础命令

|  |
| --- |
| Mysql不区分大小写。  版本-当前时间：  select version(),current\_date;  创建数据库：  create database mydb  使用数据库：  use mydb  创建表：  create table A\_XSJBXXB(  xh int primary key auto\_increment,##主键自增  xm varchar(20),  sr Date,  jtdz varchar(100),  bjdm varchar(30),  lxdh varchar(50),  ssbh varchar(50)  );  修改表明：  rename table 旧表名 to 新表名;  现实表结构：  desc user  现实创建表语句：  show create table user;  加载文件数据到表中：  load data local infile '/root/mydb.data' into table user;  数据之间用tab键隔开，为空的数据用\N代替；如下：    删除表数据：  Delete from user;  更新数据：  update user set name='xxx' where id =5;  去除重复字段的行：id/name值都重复的不在返回。  select distinct id,name from user;  排序：order by xxx desc##降序  select \* from user order by id desc;  多列排序：  select \* from user order by id desc,name asc; |

## 清空表

|  |
| --- |
| truncate table;//先清空表，再重建表，其中的自增字段会重新计算。 |

## 已有表数据插入到新表

|  |
| --- |
| -- 蠕虫复制  insert into my\_copy select \* from my\_collate\_bin; |

## 多表查询

|  |
| --- |
| 笛卡尔积：x(三条)，y(两条 )，x中的每条记录和y中每条记录的关联，即为：x \* y =2 x 3  **内连接：交集数据**    Select \* from table1 inner join table2 on 条件  **左外连接：左边的全部和两表的交集**    Select \* from talbe1 left join table2 on 条件  右外连接：右边表全部和两表交集。  Select \* from talbe1 right join table2 on 条件##  全外连接：左连接和右连接的合集，但会去除重复。**Mysql不支持**。  Select \* from talbe1 full outer join table2 on 条件 |

## Union和union all

|  |
| --- |
| Union 联合查询（但会去除重复数据）  Select \* from A Union select \* from B  Union all 不去除重复 |

## Case when

|  |
| --- |
| 语法如下：  Case sex  When ‘1’ then ‘男’  When ‘0’ then ‘女’  Else ‘其他’ end  如下例子：  select  case name  when 'space' then 'wo'  when 'ccc' then 'ta'  end as flag  from user;  或者  select  case  when id > 4 then 'cha'  when id < 3 then 'lihai'  end as flag  from user; |

## 日期函数

|  |
| --- |
| 日期：  select curdate();  获取年：  select YEAR(curdate());  select year('2019-09-09'）；  获取月:  select month(curdate());  获取日：  select day(curdate());  时间差值：  select timestampdiff(year,'1989-10-10',curdate());###year为相差的单位，'1989-10-10‘被减数  select timestampdiff(month,'1989-10-10',curdate()); |

## 聚合函数

|  |
| --- |
| * Count:统计**不为NULL**的个数 * Sum:列中之和,如果sum进行多列求和，其中一列有NULL那么该行其他列也不被sum计算在内。 |

## Null值判断

|  |
| --- |
| Is null/is not null  select \* from user where name is not null;  select \* from user where name is null; |
|  |

## 主键冲突处理

|  |
| --- |
| 主键已经存在再去插入即为主键冲突  主键冲突-更新：  -- 当主键冲突的时候，进行更新操作  insert into my\_class values ('PM3527','B315')  -- 冲突处理  on duplicate key update  -- 更新主键值  room = 'B315';  主键冲突-替换：  -- 当主键冲突的时候，进行替换操作  replace into my\_class values ('PM3528','B215'); |

## 索引

|  |
| --- |
| 更具某种算法将已有的数据（后边新增的数据）建立一个文件，这个文件能够快速匹配数据，找到对应记录（几乎所有的索引都建立在字段之上的）。提升查询效率，约束数据的有效性，但索引文件会占用磁盘空间。降低更新表的速度（需更新索引文件）  **MySQL 中提供了多种索引，包括：**   * **主键索引primary key**      * **唯一键索引unique key**   创建唯一索引：create unique index 索引名 on 表（列）  添加索引：alter table 表名add unique索引名(字段);  创建表时指定：  create table index\_test2(  id int not null,  Unique id\_index(id)  );   * **全文索引fulltext index**      * **普通索引index**   创建索引：no\_index 为索引名，no为索引对象的字段  create index no\_index on index\_tab(no);  已有的表添加索引：alter table 表名add index 索引名(字段);  alter table index\_test add index id\_index(id);  创建时指定索引：  create table index\_test2(  id int not null,  index id\_index(id)  );  删除索引：drop index 索引名 on 表  drop index id\_index on index\_test2;  查询索引：  show index from test.tab1; |

## 事务

|  |
| --- |
| 保证一组数据库操作的一致性操作（要么成功/失败）  事务必须满足（ACID）    **事务控制语句：**  Begin或start transaction 显示开启事务  Commit：提交事务  Rollback：事务失败时回滚操作。  Set transaction :设置隔离级别  **Mysql事务处理：**   * **使用 begin-commit-rollback** * **设置mysql的自动提交模式：set AUTOCOMMIT=0禁止自动提交，set AUTOCOMMIT=1开启自动提交。**   AUTOCOMMIT=1开启后每条操作后会自动提交事务，除非显示开启事务。  AUTOCOMMIT=0 则系统会自动开启，但需要手动提交事务，显示提交。  查看：show variables like 'autocommit'; |

## 函数

|  |
| --- |
| 定义函数：随机产生n个字符串。  DELIMITER $  create function randstr(n int) returns varchar(255)  begin  declare all\_str varchar(100) default 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';  declare i int default 0;  declare return\_str varchar(255) default '';  while i<n  do  set return\_str=concat(return\_str,substring(all\_str,floor(1+rand()\*52),1));  set i = i + 1;  end while;  return return\_str;  end $  调用：select randstr(2); |

## 存储过程

|  |
| --- |
| create procedure 过程名([参数列表])  begin  -- 过程体  end  显示存储过程：show procedure status + [like 'pattern'];  过程创建语句：show create procedure + 过程名;  调用过程：call + 过程名;  删除过程：drop procedure + 过程名;    In 可传入常量或变量，是变量的话，变量的值就相当于复制了一份传入到存储过程中，所以执行完后变量不会改变。  Out:只传入变量，并且变量的值被初始化清空的，存储过程中如果修改了变量，过程结束后会影响变量的值。  Inout:和out类似但不清空值。 |

## 数据库备份恢复

|  |
| --- |
| 备份数据库：mydb为数据库名。  mysqldump -uroot -proot **mydb**>~/mydb.sql  数据库恢复：必须先创建数据库  Myslq -uroot -proot mydb2<~/mydb.sql |

## Mysql注意

|  |
| --- |
| * MySQL 字符串比较 不区分大小写。 |

# 数据库设计

|  |
| --- |
| 数据库设计步骤：   * **需求分析** * 数据是什么 * 数据有哪些属性 * 数据和属性各自的特点有哪些。 * **逻辑设计**   ER图逻辑建模   * **物理设计**   根据数据特点选择数据库   * **维护优化** * 新需求进行建表 * 索引优化 * 大表拆分 |
|  |

## 数据库设计遵循的范式

解决数据库存储和优化问题，减少冗余。分层结构，共6层，一层比一层严格，1NF是基础。下一层必须满足上一层。范式只是从存储角度考虑，并没有从效率方面考虑，所以范式只能作为参考。一般满足**三范式**就可以

|  |
| --- |
| **第一范式（原子性）**：数据不可分割，（字段不可分割，具有原子性）  **第二范式（唯一性）**：解决表设计中不允许出现部分依赖。  必须有主键，主键决定唯一性，其他的非主键必须依赖于主键，联合主键（不能依赖部分）  如下：学生姓名只依赖（和xx有关）学生编号，但这一行数据唯一性需要（学生编号和教师编号共同决定）。一行数据只描述以件事儿。    第三范式：满足1NF和2NF,依赖某一个字段，而该字段依赖主键，的这种间接依赖，  应该避免**传递依赖的问题** |

# 数据库原理

|  |
| --- |
|  |

## 连接器

|  |
| --- |
| 连接数据库，首先需要登录连接。连接器负责跟客户端**建立连接**、**获取权限**、**维持和管理连接**。  mysql -h$ip -P$port -u$user -p |

## 查询缓存

|  |
| --- |
| Mysql8.0之后取消查询缓存。  显示的指出从缓存中查询：  select SQL\_CACHE \* from T where ID=10；  连接登录后拿到的查询语句会先在查询缓存中查找。如果有则返回给客户端，无则继续执行后续步骤。  **但是大多数情况下我会建议你不要使用查询缓存，为什么呢？因为查询缓存往往弊大于利。**  查询缓存在一张表**更新（delete,update,insert）**时，这张表的所有查询缓存就会**失效（清空**），所以存储的缓存往往无法使用就被清空.  但可以给静态表，并且经常使用的表做查询缓存。 |

## 分析器

|  |
| --- |
| * 词法解析（输入的字符串，代表什么的解析工作） * 语法解析（判断收入的是否符合语法规则等等。。。）如下： |

## 优化器

|  |
| --- |
| 经过**优化器，mysql**知道你需要做什么，在执行前还要经过优化器处理。如：   * 决定使用哪个索引 * Join时决定各个表的连接顺序 |

## 执行器

|  |
| --- |
| 执行前，先判断你对该表有没有**执行查询的权限**    如无索引：    有索引：和上边差不多 |

## 更新语句的执行过程

|  |
| --- |
| 如：update T set c=c+1 where ID=2;    如果日志文件被写满则必须去更新到磁盘上，并清除日志中已经到磁盘的内容。    类似一个环形队列。  有了redo log，InnoDB就可以保证即使数据库发生异常重启，之前提交的记录都不会丢失，这个能力称为**crash-safe**。  redo log是InnoDB引擎特有的日志。  Binlog日志：是mysql server层实现的日志，记录所有逻辑操作。  这两种日志有以下三点不同。   * redo log是InnoDB引擎特有的；binlog是MySQL的Server层实现的，所有引擎都可以使用。 * redo log是物理日志，记录的是“在某个数据页上做了什么修改”；binlog是逻辑日志，记录的是这个语句的原始逻辑，比如“给ID=2这一行的c字段加1 ”。 * redo log是循环写的，空间固定会用完；binlog是可以追加写入的。“追加写”是指binlog文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。   IMG_256  最后三步看上去有点“绕”，将redo log的写入拆成了两个步骤：prepare和commit，这就是"两阶段提交"。  两个阶段：避免使用日志恢复的数据和数据库本身的不一致。如果不使用“两阶段提交”，那么数据库的状态就有可能和用它的日志恢复出来的库的状态不一致。    redo log用于保证crash-safe能力。innodb\_flush\_log\_at\_trx\_commit这个参数设置成1的时候，表示每次事务的redo log都直接持久化到磁盘。这个参数我建议你设置成1，这样可以保证MySQL异常重启之后数据不丢失。  sync\_binlog这个参数设置成1的时候，表示每次事务的binlog都持久化到磁盘。这个参数我也建议你设置成1，这样可以保证MySQL异常重启之后binlog不丢失。 |

## 事务

|  |
| --- |
| 事务就是要保证一组数据库操作的原子性，要么成功，要么失败。  Mysql中事务实在**引擎层**实现的。  Mysql的原生引擎：MyISAM不支持事务，这也是InnoDB取代它的原因。  事务的特点：ACID（即原子性、一致性、隔离性、持久性）。  SQL标准的事务隔离级别包括：   * **读未提交**（read uncommitted）：   一个事务还没提交时，它做的变更就能被别的事务看到  另一个事务可以看到其他没有提交的改动。   * **读提交**（read committed）：   一个事务提交之后，它做的变更才会被其他事务看到   * **可重复读**（repeatable read）：   一个事务执行过程中看到的数据，总是跟这个事务在启动时看到的数据是一致的。  启动时数据是什么，事务执行过程中数据就是什么。   * **串行化**（serializable ）：   顾名思义是对于同一行记录，“写”会加“写锁”，“读”会加“读锁”。当出现读写锁冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行。  如下例子：  mysql> create table T(c int) engine=InnoDB;  insert into T(c) values(1);      设置隔离级别：transaction-isolation 参数设置 |

## 索引

|  |
| --- |
| Mysql索引是在**存储引擎层**实现的。表都是根据主键顺序以索引的形式存放的  InnoDB 的索引模型（如：哈希表，有序数组，二叉树（多叉树））：**B+树索引模型**    Mysql innoDB引擎 是索引组织表（也就是数据以索引形式存储的，一般行数据存储在**主键索引（**聚集索引**）对应key的叶子节点**上。）如果没有主键呢？如下：  1.如果声明了主键(primary key)，则这个列会被做为聚集索引  2.如果没有声明主键，则会用一个唯一且不为空的索引列做为主键，成为此表的聚集索引  3.上面二个条件都不满足，InnoDB会自己产生一个虚拟的聚集索引。  索引创建和使用较好的原则：   * 覆盖索引：索引已经覆盖了查询的结果。（不需要回表） * 最左前缀：最先匹配左边字段（字符串前边：’xx%’）,这样这个索引的高效检索就会被用到，但如果如：index(a,b),只使用b,去查询b则也使用了索引，但**需要遍历整个索引，效率较低。** * 索引下推：mysql5.6之后推出的，如：   Index(name,age)  Select \* from tab where name like ‘1%’ and age =10 and sex =1  如上 mysql会 找到‘1%’的数据 再找到 age =10的数据 再回表判断其他条件符合的数据。 |

# Mysql优化

## 支持的引擎

|  |
| --- |
| Mysql引擎：  innoDB:事务优先，（行级锁，适合高并发）  myISAM:性能优先（表级锁,不支持事务）  show engines;    Mysql5.4之后默认 innodb |

## 指定引擎

|  |
| --- |
| create table test.tab1(  id int primary key auto\_increment,  name varchar(10)  )engine=MyISAM;  制定表的引擎 |

## SQL优化

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 优化重点在于优化索引。  索引（index）相当与目录    索引优点：   * 提高查询效率，降低IO的使用次数。 * 降低CPU使用率（索引本身就是排序的树结构，因此如：order by 操作 不需要额外排序）   索引弊端：   * 索引文件会占据磁盘 * 如下情况不适应使用索引：少量数据，频繁更新字段（索引文件频繁改动），很少使用的字段 * 降低（增删改）的效率。   B树：数据都存储在叶节点，其他只是用来表示 数据范围（3-10）       * 单值索引：单列，如：age,一个表可以有多个单值索引 * 唯一索引：不能重复。 * 复合索引：多个列构成索引，相当于有二级目录，如：name-age,先按照name查，如果有多个，在按照age查。   创建索引方式一：  create index index\_name on tab(id);##单值索引  Create unique index index\_name2 on tab(id);##唯一索引  Create index index\_name on tab(name,age);##复合所用  主键自动添加索引。  SQL性能问题：   * 分析sql执行计划 * Mysql的优化器可能会干扰你的优化操作（它认为你的sql还需要优化）   Sql执行计划：explain + SQL语句  explain select \* from test where id =1;  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+  | id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+  | 1 | SIMPLE | test | const | PRIMARY | PRIMARY | 4 | const | 1 | NULL |  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | select\_type | Table | Type | possible\_keys | Key | Key\_len | Ref | Rows | Extra | | 查询类型 | 表名 | 类型 | 预测用到的索引 | 实际用的索引 | 索引长度 | 表和表之间的引用 | 通过索引查到的数据量 | 额外信息 | |

## SQL语句执行计划分析-实例

|  |
| --- |
| 构建三张表：  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 teacher\_card(tcid int(3),tcdesc varchar(200));  插入数据后执行：  explain select t.\* from teacher t,course c,teacher\_card tc where t.tid=c.tid and t.tcid=tc.tcid and (c.cid =2 or tc.tcid =3);    **如上ID相同**，执行顺序由上往下，如果给第一条中的t表加入几条数据，上如顺序会改变  数据小的表会优先执行，这样 中间结果数据少，占用内存少。    **如果ID不同：id越大越优先执行。先查询内层sql再查询外层**  **如下：**  explain select tc.\* from  teacher\_card tc where tc.tcid =(  select t.tcid from teacher t where t.tid =(  select c.tid from course c where c.cname='sql'  )  );    **如果id有相同有不同：先执行id大的，id相同 从上往下顺序执行。**  explain select tc.\* from teacher\_card tc where tc.tcid =(select t.tcid from teacher t,course c where t.tid=c.tid and c.cname='sql' );    **select\_type：查询类型**   * **PRIMARY:包含子查询中的最外层查询（主查询）** * **SUBQUERY：包含子查询中的 子查询（非最外层查询）** * **SIMPLE：不包含子查询，union** * **Derived:衍生查询（使用到了临时表）** * From子查询中只有一张表   如：explain select cc.cname from (select c.\* from course c where c.cid in (1,2,3)) cc;     * From子查询中，如果有table1 union table2,则table1就是derived表，table2为union表。   如：explain select cc.cname from (select c.\* from course c where c.cid in (1,2) union select c.\* from course c where c.cid in (3)) cc;    **Type**:索引类型，类型，如下较为常见（必须有索引才能去优化type）  System > const > eq\_ref > ref > range > index > all  **越往左性能越高**：System，const 理性情况，一般达不到。实际达到：ref ，range   * System:如果表满足：只有一条数据的系统表，或一条数据的衍生表。（基本达不到） * Const：仅仅能查出一条数据，用于primary key或unique（索引必须这两个） * eq\_ref：对于每个索引键的查询(查询索引键)，返回的数据必须唯一（有且只有一个，不能多，也不能少。）常见于：主键索引，唯一键索引（**一般达不到**）   如：A(id,name,bid),B(bid,name)  给A表bid加唯一索引，B表bid加主键索引。如下查询  select a.bid from A a,B b where a.bid = b.bid;A表中数据必须返回全部，有一个没有就达不到。   * ref ：对于每个索引键的查询(查询索引键)，返回多个（0，多个）   如：给某字段加索引，按照索引查询  explain select \* from course where tid = 1;   * range ：范围查询(根据索引字段查询)，如：between,in ,>,<,>=   explain select \* from course where tid <3;  注意：in有时会失效。   * Index:遍历索引树查询，通常发生在查询结果只包含索引字段时 * All：全表扫描，没用到任何索引。   **Key\_len：**索引长度，由于判断符合索引是否被完全使用。（某个索引没用则长度少了）  （utf8：一个字符占3个字节,如果字段可以为null,要多一个字节来表示null,如果为可变长多占2个字节，如：varchar）    **Ref：当前表参照的字段，如：**  Select \* from tab1 t1,tab2 t2 where t1.t2id=t2.id and t2.id=1  如果是其他字段，ref为如：t2.id,如果是常量为：const  **Rows:**这个数表示mysql要遍历多少数据才能找到，表示MySQL根据表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，**在innodb上可能是不准确的.**  **根据索引查询的行数**  **Extra：额外的信息。**   * **Using filesort:性能消耗大，需要额外的一次排序。（常见在order by语句）**   **如：**create table(  a int,  b int,  c int,  index a\_index(a),  Index b\_index(b),  index c\_index(c),  )  如下查询需要额外的一次排序  Select \* from table where a=1 order by **b**  符合索引：如 符合索引（a,b,c）最佳左前缀，不能跨列  Select \* from tab where a order by c###跨过了b  如下就不是using filesort了：  Select \* from tab where a order by b  **避免：**  单索引：where 哪些字段就order by 哪些字段  符合索引：按照符合索引顺序 使用。   * **Using temporary:性能损耗大，用的临时表（一般出现在group by的sql）**     Sql解析过程顺序：  from ....on ..join...where...group by....having...selcet ....order by..limit   * **Using index:性能提升了，索引覆盖。（只需要从索引中查询，不需要回表。）**      * **Using where:在查找使用索引的情况下，需要回表去查询所需的数据** * **Impossible WHERE:不可能的where条件，如下：**   explain select tid from course where tid =1 and tid = 0;##又等于1又等于0  如下优化实例： |

## 优化案例-单表

|  |
| --- |
| 单表优化：  准备表：  create table book(  bid int primary key,  bname varchar(20) not null,  authorid int not null,  publicid int not null,  typeid int not null  )  insert into book values(1,'java',1,1,1);  insert into book values(2,'c',2,2,1);  insert into book values(3,'cad',3,3,2);  insert into book values(4,'语文',4,4,3);  如下SQL:  select bid from book where typeid in(2,3) and authorid=3 order by typeid desc;  创建索引：（如复合索引，（typeid,authorid））**覆盖索引**  查看sql执行计划：Extra（using where;using index）    但，type：index级别还可以优化：in 可能 失效，所以将typeid放后边  重建索引：（authorid，typeid）（实际不能按照一个sql去优化，应该将查询概率多的字段放在前边。）  优化后查询：  explain select bid from book where authorid=3 and typeid in(2,3) order by typeid desc;    总结：   * 索引不能跨列使用，**最佳左前缀** * 将in尽量放在最后，防止失效 * 覆盖索引 |

## 优化案例-多表

|  |
| --- |
| 如下创建两个表：  create table teacher2(  tid int primary key,  cid int not null  );  insert into teacher2 values(1,2);  insert into teacher2 values(2,1);  insert into teacher2 values(3,3);  create table course2(  cid int,  cname varchar(20)  );  insert into course2 values(1,'java');  insert into course2 values(2,'python');  insert into course2 values(3,'kotlin');  如下左外连接：  select \* from teacher2 t left join course2 c on t.cid = c.cid where c.cname='java';  一般将小表放前边：  如：teacher2(3条数据)，course2(5条数据)    ###虽然都是3\*5=15条数据但外层循环少，效率更高。  For(小表){  For(大表){  }  }   * 所以就是小表驱动大表 * 经常使用的表加索引，如上（t.cid=c.cid）小表的被循环多，经常使用，所以在其上给cid建立索引。   如下优化：  create index teacher2\_cid on teacher2(cid);  create index course2\_cname on course2(cname); |

## 避免索引失效

|  |
| --- |
| * 最好遵循最左前缀 * 尽量使用全部索引（index(a,b,c)）都使用。 * 不要在索引上进行操作（计算，函数，类型转换）   如下：select...where a.x\*3 = ,这样a.x索引会失效。   * 复合索引使用（！=，<>,is null,is not null）自身和右侧索引就会失效。 * 复合索引 如遇到：> < 等范围查询，可能让右侧索引失效（本身一般有效）。（优化器会衡量是否使用索引的效率问题） * Like 尽量以常量开头，否则索引失效。如：‘%xxx’,如果非得‘%xxx%’可以使用索引覆盖提升性能。 * 尽量不要使用类型转换，否则索引失效。   如下：cname为varchar类型，但给其int类型，需要类型转换，让索引失效了。  explain select \* from course2 where cname=123   * Or可能使索引失效，甚至 or左侧的索引也失效。 |

## 其他优化方式

|  |
| --- |
| * Exist和in的选择   Select \* from tab where exist (子查询)  Select \* from tab where id in (子查询)  如果主查询数据量大，则用in效率大于exist  如果子查询数据量大，则用exist效率大于in     * Order by 优化   多出的排序会出现：using filesort(两种算法：双路排序（多次扫描IO），和单路排序)  Mysql4.1之前为双路排序。1.先扫描排序字段，排序。2.再扫描其他需要的字段  Mysql4.1之后为单路排序。扫描所有需要字段。(不一定一次IO,可能多次)    设置buffer: set max\_length\_for\_sort\_data =1024 (单位字节)  如果max\_length\_for\_sort\_data太小，则mysql会自动从单路切换到双路。需要排序的字段总量大于buffer  **提高order by 查询效率：**   * 使用单路，双路；调整buffer容量 * 避免 select \*   \*需要计算有哪些列，并且\*也不能覆盖索引。   * 尽量保证所有排序字段排序的一致性。全asc或全desc |

## Sql排查-慢查询日志

|  |
| --- |
| Mysql提供的一种日志，用于记录某个操作的响应时间超过阈值的sql,默认关闭的。  查看是否开启了慢查询日志：  show variables like '%slow\_query\_log%';  +---------------------+---------------------------------+  | Variable\_name | Value |  +---------------------+---------------------------------+  | slow\_query\_log | OFF |  | slow\_query\_log\_file | /var/lib/mysql/master2-slow.log |  +---------------------+---------------------------------+  临时开启：set global slow\_query\_log=1  永久开启：/etc/my.cnf    阈值的设置：show variables like '%long\_query\_time%';  设置这个阈值，如上一样临时设置：set global long\_query\_time=5,设置后需要重新登录。  查看慢查询的sql(超过阈值的sql个数):  show global status like '%slow\_queries%';    具体sql需要在日志文件中查找  可以使用mysql内置工具方便查询： mysqldumpslow |

## 分析海量数据

|  |
| --- |
| 准备数据：  create database testdata;  use testdata;  create table dept(  dno int(5) primary key default 0,  dname varchar(20) not null default '',  local varchar(30) not null default ''  )engine=innodb default charset=utf8;  create table emp(  eid int(5) primary key,  ename varchar(20) not null default '',  job varchar(20) not null default '',  deptno int(5) not null default 0  )engine=innodb default charset=utf8;  存储过程插入数据   * **profiles**   打开 前会话资源的消耗情况（如：sql执行时间），从打开之后的执行的语句。  查看是否开启：show variables like '%profiling%';  开启（临时）：set profiling =on;  Show profiles；如下：  +----------+------------+-----------------------------------+  | Query\_ID | Duration | Query |  +----------+------------+-----------------------------------+  | 1 | 0.00369425 | show variables like '%profiling%' |  | 2 | 0.71788800 | select count(1) from emp |  | 3 | 0.00162650 | select count(1) from dept |  +----------+------------+-----------------------------------+   * **SQL诊断：**   如上只能现实大概时间，没有具体详尽时间。可以如下：  show profile all for query 2;##2为上边的query\_id    如下可以只查询某些关键字段：  show profile cpu,block io for query 2;     * **全局查询日志：**记录开启之后的全部sql语句（调优或开发时开启）   查看是否开启：show variables like '%general\_log%'  开启：  set global general\_log=1;  set global log\_output='table';##写入表中，**默认只放入文件中**。  会记录在mysql.general\_log表中： |

## 锁机制

|  |
| --- |
| **更具操作类型分**：   * 读锁（共享锁）：同一个数据加锁，多个操作可以同时进行，互不干扰。 * 写锁（互斥锁）：当前写操作没有完毕，则无法进行其他的写操作，读操作。   **更具范围分**：   * 表锁：给整个表加锁.如：MyISAM引擎。开销小，加锁快，容易发生锁冲突。并发度低。不会出现死锁。 * 行锁：给一行数据加锁。如：innodb引擎。开销大，加锁慢，容易出现死锁，不易冲突，并发度高。 * 页锁   **加表锁**：  Lock table 表 read/write(读锁或写锁)  释放锁：unlock tables;或者是用 提交事务commit或callback释放锁。  **如果给表A加读锁：**   * 加锁的当**前会话**只能对A表读，其他操作都行。 * 其他会话，对其他表可以做任意操作，但对A表只能读，写只能等A表锁释放才可以。   **如果给表A加写锁：**   * 加锁的当前会话可以对A表进行任何操作（CRUD），但不能操作其他表。 * 其他会话需要等待释放锁后，才能对A表（CRUD）,其他表正常操作。     查看哪些表被锁了：show open tables;  In\_use=1则表示表被锁，show open tables where in\_use>0;来查询  查看表锁的严重程度：show status like 'table%';  ----------------------------+--------+  | Variable\_name | Value |  +----------------------------+--------+  | Table\_locks\_immediate | 800367 |  | Table\_locks\_waited | 0 |    **行锁：innodb默认为行锁。（会给where后的范围加锁，不管有没有。）**  如果某一会话对row1行进行（增删改）操作，该行被加锁，直到事务提交或回滚，行锁被释放。如：commit;rollback  其他会话需等待该行锁被释放，才能对该行进行（增删改）.  **注意：**   * **如果没有索引，行锁会转化为表锁。Name非索引。或索引失效。**   如：update tab set x = ‘’ where name=’xxx’;   * **间隙锁：**   **如：**update tab set age=1 where id >1 and id <8;  会给where后的区间数据加锁，也包括中间id不存在的行（间隙），这些id其他会话insert时会等待锁的释放。  行锁分析：show status like '%innodb\_row\_lock%';  +-------------------------------+-------+  | Variable\_name | Value |  +-------------------------------+-------+  | Innodb\_row\_lock\_current\_waits | 0 |  | Innodb\_row\_lock\_time | 63054 |  | Innodb\_row\_lock\_time\_avg | 12610 |  | Innodb\_row\_lock\_time\_max | 30013 |  | Innodb\_row\_lock\_waits | 5 |  +-------------------------------+-------+    给select 加锁：select \* from line\_tab for update;  只有select释放锁了后，其他会话才能进行（增删该）操作。 |
|  |

## Mysql主从（集群）

|  |
| --- |
| 准备两个mysql    配置前先关闭防火墙，允许mysql远程访问权限等。  Master(主):配置后重启mysql  [mysqld]  server-id=1  log-bin=/var/lib/mysql/mysql-bin##开启binlog日志  binlog-ignore-db=mysql###忽略同步的数据库  binlog-do-db=test###同步哪个数据库  主数据库中添加一个同步账号：  grant replication slave,reload,super on \*.\* to 'root'@'192.168.134.99' identified by 'root';  flush privileges;  查看主数据库的状态：  show master status;    从节点配置：保存后重启mysql  server-id=2  log-bin=/var/lib/mysql/mysql-bin  replicate-do-db=test  给从节点指定master  CHANGE MASTER TO  MASTER\_HOST='192.168.134.1',  MASTER\_USER='root',  MASTER\_PASSWORD='root',  MASTER\_PORT=3306,  MASTER\_LOG\_FILE='mysql-bin.000001',  MASTER\_LOG\_POS=120;  flush privileges;  MASTER\_LOG\_FILE和MASTER\_LOG\_POS为主库中查找的状态，文件和位置点  在从节点开启：start slave;  查看状态： show slave status \G######\G让列竖着显示  Slave\_IO\_Running: Connecting  Slave\_SQL\_Running: Yes  如上两列如果都显示YES则正确，上边错误由于从节点无法登录到master导致。  将主节点配置文件中：bind-address = 127.0.0.1（注释掉）重启主节点并查看    修改从节点：  stop slave;  CHANGE MASTER TO  MASTER\_LOG\_FILE='mysql-bin.000002';  start slave;  show slave status \G  如下正确：    测试：  在主节点的test库下创建表，插入数据。在从节点mysql中查看test库中是否有该表。 |

## 空