# 第九章 数据库

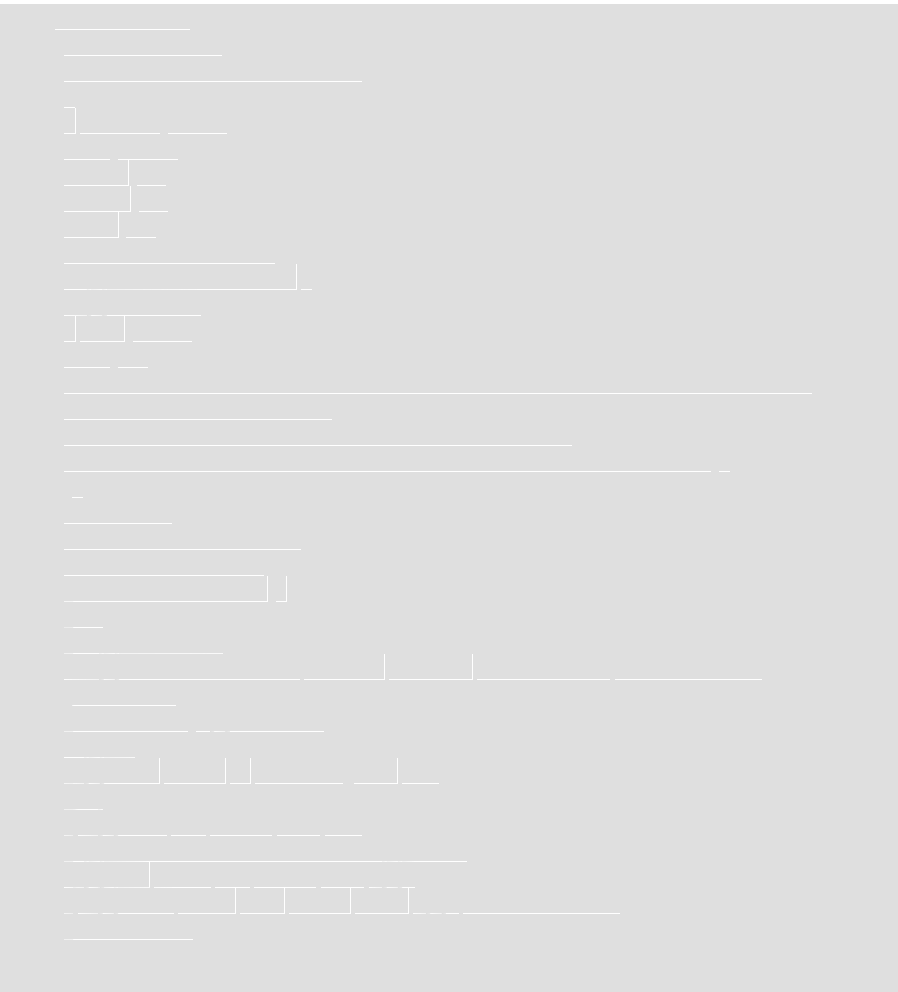
## 一．Mysql

#### **Python 中操作Mysql 步骤（2018-3-31-sxd)**

|  |
| --- |
|  |
|  |  |

代码实现：

|  |
| --- |
|  |
|  |  |

1. user：连接的用户名
2. password：连接的密码
3. charset:通信采用的编码方式，推荐采用utf8 12. '''
4. ''' connection 对象方法
5. close() 关闭连接
6. commit() 提 交
7. rollback() 回 滚
8. cursor() 返 回
9. cursor 对象，用于执行sql 语句
10. 例如：select,insert,update,delete '''
11. cs1 = conn.cursor()
12. ''' cursor 对象方法
13. close() 关 闭
14. 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，
15. 也可以执行create、alter、drop 等语句
16. 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
17. 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 ''' 27. '''
18. cursor 对象属性
19. rowcount 只读属性，表示最近一次
20. execute()执行后受影响的行数
21. connection 获得当前连接对象 '''
22. #例子
23. #创建 Connection 连接
24. conn = connect(host='localhost', port=3306, user='root', password='mysql', database='python1',
25. charset='utf8')
26. #获得 Cursor 对象 cs = conn.cursor()
27. # 更 新 #
28. sql = 'update students set name="刘邦" where id=6'
29. #删除
30. # sql = 'delete from students where id=6'
31. #执行 select 语句，并返回受影响的行数：查询一条学生数据
32. sql = 'select id,name from students where id = 7'
33. # sql = 'SELECT id,name FROM students WHERE id = 7' count=cs.execute(sql)
34. #打印受影响的行数
35. print count

#### **SQL 的select 语句完整的执行顺序**

SQL Select 语句完整的执行顺序：

188

1、from 子句组装来自不同数据源的数据；

2、where 子句基于指定的条件对记录行进行筛选； 3、group by 子句将数据划分为多个分组；

4、使用聚集函数进行计算；

5、使用 having 子句筛选分组； 6、计算所有的表达式；

7、select 的字段；

8、使用 order by 对结果集进行排序。

SQL 语言不同于其他编程语言的最明显特征是处理代码的顺序。在大多数据库语言中，代码按编码顺序被处理。但在 SQL 语句中，第一个被处理的子句式 FROM，而不是第一出现的 SELECT。SQL 查询处理的步骤序号：

1. FROM <left\_table>
2. <join\_type> JOIN <right\_table>
3. ON <join\_condition>
4. WHERE <where\_condition>
5. GROUP BY <group\_by\_list>
6. WITH {CUBE | ROLLUP}
7. HAVING <having\_condition>
8. SELECT
9. DISTINCT
10. ORDER BY <order\_by\_list>
11. <TOP\_specification> <select\_list>

以上每个步骤都会产生一个虚拟表，该虚拟表被用作下一个步骤的输入。这些虚拟表对调用者(客户端应用程序或者外部查询)不可用。只有最后一步生成的表才会会给调用者。如果没有在查询中指定某一个子句，将跳过相应的步骤。

逻辑查询处理阶段简介：

1、 FROM：对 FROM 子句中的前两个表执行笛卡尔积(交叉联接)，生成虚拟表 VT1。

2、 ON：对 VT1 应用 ON 筛选器，只有那些使为真才被插入到 TV2。

3、 OUTER (JOIN):如果指定了 OUTER JOIN(相对于 CROSS JOIN 或 INNER JOIN)，保

留表中未找到匹配的行将作为外部行添加到 VT2，生成 TV3。如果 FROM 子句包含两个以上的表，则对上一个联接生成的结果表和下一个表重复执行步骤 1 到步骤 3，直到处理完所有的表位置。

4、 WHERE：对 TV3 应用 WHERE 筛选器，只有使为 true 的行才插入 TV4。

5、 GROUP BY：按 GROUP BY 子句中的列列表对 TV4 中的行进行分组，生成 TV5。

6、 CUTE|ROLLUP：把超组插入 VT5，生成 VT6。

7、 HAVING：对 VT6 应用 HAVING 筛选器，只有使为 true 的组插入到 VT7。

8、 SELECT：处理 SELECT 列表，产生 VT8。

9、 DISTINCT：将重复的行从 VT8 中删除，产品 VT9。

10、 ORDER BY：将VT9 中的行按ORDER BY 子句中的列列表顺序，生成一个游标(VC10)。

11、 TOP：从 VC10 的开始处选择指定数量或比例的行，生成表 TV11，并返回给调用者。where 子句中的条件书写顺序

#### **说一下Mysql 数据库存储的原理？**

储存过程是一个可编程的函数，它在数据库中创建并保存。它可以有 SQL 语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。它允许控制数据的访问方式。存储过程通常有以下优点：

1、存储过程能实现较快的执行速度

2、存储过程允许标准组件是编程。

3、存储过程可以用流程控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

4、存储过程可被作为一种安全机制来充分利用。

5、存储过程能够减少网络流量

#### **4.** **事务的特性？**

1、原子性(Atomicity)：事务中的全部操作在数据库中是不可分割的，要么全部完成，要么均不执行。

2、一致性(Consistency)：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致。

3、隔离性(Isolation)：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。

4、持久性(Durability)：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障

#### **5.** **数据库索引？**

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用 B\_TREE。B\_TREE 索引加速了数据访问，因为存储引擎不会再去扫描整张表得到需要的数据；相反，它从根节点开始，根节点保存了子节点的指针，存储引擎会根据指针快速寻找数据。

#### **数据库怎么优化查询效率？**

1、储存引擎选择：如果数据表需要事务处理，应该考虑使用 InnoDB，因为它完全符合 ACID 特性。如果不需要事务处理，使用默认存储引擎 MyISAM 是比较明智的

2、分表分库，主从。

3、对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索

引

4、应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全

表扫描

5、应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫

描

6、应尽量避免在 where 子句中使用 or 来连接条件，如果一个字段有索引，一个字段没有索引，

将导致引擎放弃使用索引而进行全表扫描

7、Update 语句，如果只更改 1、2 个字段，不要 Update 全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志

8、对于多张大数据量（这里几百条就算大了）的表 JOIN，要先分页再 JOIN，否则逻辑读会很高， 性能很差。

#### **Mysql 集群的优缺点？**

优点：

* 99.999%的高可用性
* 快速的自动失效切换
* 灵活的分布式体系结构，没有单点故障
* 高吞吐量和低延迟
* 可扩展性强，支持在线扩容缺点：
* 存在很多限制，比如：不支持外键
* 部署、管理、配置很复杂
* 占用磁盘空间大、内存大
* 备份和恢复不方便
* 重启的时候，数据节点将数据 load 到内存需要很长的时间

#### **你用的Mysql 是哪个引擎，各引擎之间有什么区别？**

主要 MyISAM 与 InnoDB 两个引擎，其主要区别如下：

InnoDB 支持事务，MyISAM 不支持，这一点是非常之重要。事务是一种高级的处理方式，如在一些列增删改中只要哪个出错还可以回滚还原，而 MyISAM 就不可以了；

MyISAM 适合查询以及插入为主的应用，InnoDB 适合频繁修改以及涉及到安全性较高的应用； InnoDB 支持外键，MyISAM 不支持；

MyISAM 是默认引擎，InnoDB 需要指定； InnoDB 不支持 FULLTEXT 类型的索引；

InnoDB 中不保存表的行数，如 select count() from table 时，InnoDB；需要扫描一遍整个表来计算有多少行，但是 MyISAM 只要简单的读出保存好的行数即可。注意的是，当 count()语句包含where 条件时 MyISAM 也需要扫描整个表；

对于自增长的字段，InnoDB 中必须包含只有该字段的索引，但是在 MyISAM 表中可以和其他字段一起建立联合索引；清空整个表时，InnoDB 是一行一行的删除，效率非常慢。MyISAM 则会重建表；

InnoDB 支持行锁（某些情况下还是锁整表，如 update table set a=1 where user like '%lee%'

#### **9.** **数据库的优化？**

1. 优化索引、SQL 语句、分析慢查询；
2. 设计表的时候严格根据数据库的设计范式来设计数据库；
3. 使用缓存，把经常访问到的数据而且不需要经常变化的数据放在缓存中，能节约磁盘 IO
4. 优化硬件；采用 SSD，使用磁盘队列技术(RAID0,RAID1,RDID5)等
5. 采用 MySQL 内部自带的表分区技术，把数据分层不同的文件，能够提高磁盘的读取效率；
6. 垂直分表；把一些不经常读的数据放在一张表里，节约磁盘 I/O；
7. 主从分离读写；采用主从复制把数据库的读操作和写入操作分离开来；
8. 分库分表分机器（数据量特别大），主要的的原理就是数据路由；
9. 选择合适的表引擎，参数上的优化
10. 进行架构级别的缓存，静态化和分布式；
11. 不采用全文索引；
12. 采用更快的存储方式，例如 NoSQL 存储经常访问的数据\*\*。

#### **Mysql 数据库如何分区、分表？**

分表可以通过三种方式：Mysql 集群、自定义规则和 merge 存储引擎。分区有四类：

RANGE 分区：基于属于一个给定连续区间的列值，把多行分配给分区。

LIST 分区：类似于按 RANGE 分区，区别在于 LIST 分区是基于列值匹配一个离散值集合中的某个值来进行选择。

HASH 分区：基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含 MySQL 中有效的、产生非负整数值的任何表达式。

KEY 分区：类似于按 HASH 分区，区别在于 KEY 分区只支持计算一列或多列，且 MySQL 服务器提供其自身的哈希函数。必须有一列或多列包含整数值。

#### **如何对查询命令进行优化？**

1. 应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索。
2. 应尽量避免在 where 子句中对字段进行 null 值判断，避免使用!=或<>操作符，避免使用 or 连接条件，或在 where 子句中使用参数、对字段进行表达式或函数操作，否则会导致权标扫描
3. 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。
4. 使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用。
5. 很多时候可考虑用 exists 代替 in。
6. 尽量使用数字型字段。
7. 尽可能的使用 varchar/nvarchar 代替 char/nchar。

1. 任何地方都不要使用 select from t ，用具体的字段列表代替“”，不要返回用不到的任何字段。
2. 尽量使用表变量来代替临时表。
3. 避免频繁创建和删除临时表，以减少系统表资源的消耗。
4. 尽量避免使用游标，因为游标的效率较差。
5. 在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF。
6. 尽量避免大事务操作，提高系统并发能力。
7. 尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

#### **Sql 注入是如何产生的，如何防止？**

程序开发过程中不注意规范书写 sql 语句和对特殊字符进行过滤，导致客户端可以通过全局变量POST 和 GET 提交一些 sql 语句正常执行。产生 Sql 注入。下面是防止办法：

1. 过滤掉一些常见的数据库操作关键字，或者通过系统函数来进行过滤。
2. 在 PHP 配置文件中将 Register\_globals=off;设置为关闭状态
3. SQL 语句书写的时候尽量不要省略小引号(tab 键上面那个)和单引号
4. 提高数据库命名技巧，对于一些重要的字段根据程序的特点命名，取不易被猜到的
5. 对于常用的方法加以封装，避免直接暴漏 SQL 语句
6. 开启 PHP 安全模式：Safe\_mode=on;
7. 打开 magic\_quotes\_gpc 来防止 SQL 注入
8. 控制错误信息：关闭错误提示信息，将错误信息写到系统日志。
9. 使用 mysqli 或 pdo 预处理。

#### **NoSQL 和关系数据库的区别？**

1. SQL 数据存在特定结构的表中；而 NoSQL 则更加灵活和可扩展，存储方式可以省是 JSON 文档、哈希表或者其他方式。
2. 在 SQL 中，必须定义好表和字段结构后才能添加数据，例如定义表的主键(primary key)，索引(index),触发器(trigger),存储过程(stored procedure)等。表结构可以在被定义之后更新，但是如果有比较大的结构变更的话就会变得比较复杂。在 NoSQL 中，数据可以在任何时候任何地方添加，不需要先定义表。
3. SQL 中如果需要增加外部关联数据的话，规范化做法是在原表中增加一个外键，关联外部数据表。而在 NoSQL 中除了这种规范化的外部数据表做法以外，我们还能用如下的非规范化方式把外部数据直接放到原数据集中，以提高查询效率。缺点也比较明显，更新审核人数据的时候将会比较麻烦。
4. SQL 中可以使用 JOIN 表链接方式将多个关系数据表中的数据用一条简单的查询语句查询出来。NoSQL 暂未提供类似 JOIN 的查询方式对多个数据集中的数据做查询。所以大部分 NoSQL 使用非规范化的数据存储方式存储数据。
5. SQL 中不允许删除已经被使用的外部数据，而 NoSQL 中则没有这种强耦合的概念，可以随时删除任何数据。
6. SQL 中如果多张表数据需要同批次被更新，即如果其中一张表更新失败的话其他表也不能更新成功。这种场景可以通过事务来控制，可以在所有命令完成后再统一提交事务。而 NoSQL 中没有事务这个概念，每一个数据集的操作都是原子级的。
7. 在相同水平的系统设计的前提下，因为 NoSQL 中省略了 JOIN 查询的消耗，故理论上性能上是优于 SQL 的。



* 1. Mysql 数据库中怎么实现分页？

select \* from table limit (start-1)\*limit,limit; 其中 start 是页码，limit 是每页显示的条数。

* 1. sql 语句怎么看效率？

SQLServer2005-->新建一个查询-->输入语句 SELECT \* FROM Person.Contact 执行(F5)-->Ctrl+L。

* 1. 优化数据库？提高数据库的性能？

1. 对语句的优化

①用程序中，保证在实现功能的基础上，尽量减少对数据库的访问次数；

通过搜索参数，尽量减少对表的访问行数,最小化结果集，从而减轻网络负担；

②能够分开的操作尽量分开处理，提高每次的响应速度；在数据窗口使用 SQL 时，尽量把使用的索引放在选择的首列；算法的结构尽量简单；

③在查询时，不要过多地使用通配符如 SELECT \* FROM T1 语句，要用到几列就选择几列如： SELECT COL1,COL2 FROM T1；

④在可能的情况下尽量限制尽量结果集行数如：SELECT TOP 300 COL1,COL2,COL3 FROM T1,因为某些情况下用户是不需要那么多的数据的。

[⑤不要在应用中使用数据库游标](http://baike.baidu.com/view/176618.htm)，游标是非常有用的工具，但比使用常规的、面向集的 SQL 语句需要更大的开销；按照特定顺序提取数据的查找。

1. 避免使用不兼容的数据类型

例如 float 和 int、char 和 varchar、binary 和 varbinary 是不兼容的。数据类型的不兼容可能使优化器无法执行一些本来可以进行的优化操作。

198

例如:

SELECT name FROM employee WHERE salary ＞ 60000

在这条语句中,如salary 字段是money 型的,则优化器很难对其进行优化,因为 60000 是个整型数。我们应当在编程时将整型转化成为钱币型,而不要等到运行时转化。

若在查询时强制转换，查询速度会明显减慢。

1. 避免在 WHERE 子句中对字段进行函数或表达式操作**。**

若进行函数或表达式操作，将导致引擎放弃使用索引而进行全表扫描。

1. 避免使用!=或＜＞、IS NULL 或 IS NOT NULL、IN ，NOT IN 等这样的操作符
2. 尽量使用数字型字段
3. 合理使用 EXISTS,NOT EXISTS 子句。
4. 尽量避免在索引过的字符数据中，使用非打头字母搜索。
5. 分利用连接条件
6. 消除对大型表行数据的顺序存取
7. 避免困难的正规表达式
8. 使用视图加速查询
9. 能够用 BETWEEN 的就不要用 IN
10. DISTINCT 的就不用 GROUP BY
11. 部分利用索引
12. 能用 UNION ALL 就不要用 UNION
13. 不要写一些不做任何事的查询
14. 尽量不要用 SELECT INTO 语句
15. 必要时强制查询优化器使用某个索引

1. 虽然 UPDATE、DELETE 语句的写法基本固定，但是还是对 UPDATE 语句给点建议：
   1. 尽量不要修改主键字段。
   2. 当修改 VARCHAR 型字段时，尽量使用相同长度内容的值代替。
   3. 尽量最小化对于含有 UPDATE 触发器的表的 UPDATE 操作。
   4. 避免 UPDATE 将要复制到其他数据库的列。
   5. 避免 UPDATE 建有很多索引的列。
   6. 避免 UPDATE 在 WHERE 子句条件中的列。

**17.** 提取数据库中倒数 10 条数据？

select top (10) \* from table1 order by id desc。

#### **18. 数据库负载均衡**

负载均衡集群是由一组相互独立的计算机系统构成，通过常规网络或专用网络进行连接，由路由器衔接在一起，各节点相互协作、共同负载、均衡压力，对客户端来说，整个群集可以视为一台具有超高性能的独立服务器。

1、 实现原理

实现数据库的负载均衡技术，首先要有一个可以控制连接数据库的控制端。在这里，它截断了数据库和程序的直接连接，由所有的程序来访问这个中间层，然后再由中间层来访问数据库。这样， 我们就可以具体控制访问某个数据库了，然后还可以根据数据库的当前负载采取有效的均衡策略， 来调整每次连接到哪个数据库。

2、 实现多据库数据同步

对于负载均衡，最重要的就是所有服务器的数据都是实时同步的。这是一个集群所必需的，因为，如果数不据实时、不同步，那么用户从一台服务器读出的数据，就有别于从另一台服务器读出的数据，这是不能允许的。所以必须实现数据库的数据同步。这样，在查询的时候就可以有多个资源，实现均衡。比较常用的方法是 Moebius for SQL Server 集群，Moebius for SQL Server 集群采用将核心程序驻留在每个机器的数据库中的办法，这个核心程序称为 Moebius for SQL Server 中间件，主要作用是监测数据库内数据的变化并将变化的数据同步到其他数据库中。数据同步完成后客户端才会得到响应，同步过程是并发完成的，所以同步到多个数据库和同步到一个数据库的时间基本相等；另外同步的过程是在事务的环境下完成的，保证了多份数据在任何时刻数据的一致性。正因为 Moebius 中间件宿主在数据库中的创新，让中间件不但能知道数据的变化，而且知道引起数据变化的 SQL 语句，根据 SQL 语句的类型智能的采取不同的数据同步的策略以保证数据同步成本的最小化。

数据条数很少，数据内容也不大，则直接同步数据。数据条数很少，但是里面包含大数据类型， 比如文本，二进制数据等，则先对数据进行压缩然后再同步，从而减少网络带宽的占用和传输所用的时间。

数据条数很多，此时中间件会拿到造成数据变化的 SQL 语句， 然后对 SQL 语句进行解析，分析其执行计划和执行成本，并选择是同步数据还是同步 SQL 语句到其他的数据库中。此种情况应用在对表结构进行调整或者批量更改数据的时候非常有用。

3、 优缺点优点：

1. 扩展性强：当系统要更高数据库处理速度时，只要简单地增加数据库服务器就 可以得到扩展。
2. 可维护性：当某节点发生故障时，系统会自动检测故障并转移故障节点的应用，保证数据库的持续工作。

201

* 1. 安全性：因为数据会同步的多台服务器上，可以实现数据集的冗余，通过多份数据来保证安全性。另外它成功地将数据库放到了内网之中，更好地保护了数据库的安全性。
  2. 易用性：对应用来说完全透明，集群暴露出来的就是一个 IP 缺点：
     1. 不能够按照 Web 服务器的处理能力分配负载。
     2. 负载均衡器(控制端)故障，会导致整个数据库系统瘫痪。

#### **19.** **Mysql 数据库的操作?**

修改表-修改字段，重命名版：

alter table 表名 change 原名 新名 类型及约束；

alter table students change birthday birth datetime not null; 修改表-修改字段，不重名版本：

alter table 表 名 modify 列 名 类 型 和 约 束 ； alter table students modify birth date not null 全列插入：insert into 表名 values(...)

insert into students values(0,"郭靖", 1,"内蒙","2017-6"); 部分插入：值的顺序与给出的列顺序对应：

insert into students(name, birthday) values("黄蓉","2017-8"); 修改：update 表名 set 列 1=值 1，列 2=值 2.。。where

update students set gender=0, homwtown="古墓"， where id = 5; 备份：mysqldump -uroot -p 数据库名 》 python.sql,

恢复：mysql -uroot -p 数据库名 < python.sql

#### **20. 数据库的设计？**

第一范式：数据库表的每一列都是不可分割的原子数据项，即列不可拆分。

第二范式：建立在第一范式的基础上，要求数据库表中的每个实例或记录必须是可以唯一被区分的， 即唯一标识。

第三范式：建立在第二范式的基础上，任何非主属性不依赖与其他非主属性，即引用主键。

#### **21. 存储过程和函数的区别?**

相同点：存储过程和函数都是为了可重复的执行操作数据库的 sql 语句的集合。

1）存储过程和函数都是一次编译，就会被缓存起来，下次使用就直接命中已经编译好的 sql 语句， 不需要重复使用。减少网络交互，减少网络访问流量。

不同点：标识符不同，函数的标识符是 function，存储过程是 proceduce。

1. 函数中有返回值，且必须有返回值，而过程没有返回值，但是可以通过设置参数类型（in,out) 来实现多个参数或者返回值。
2. 存储函数使用 select 调用，存储过程需要使用 call 调用。
3. select 语句可以在存储过程中调用，但是除了 select..into 之外的 select 语句都不能在函数中使用。
4. 通过 in out 参数，过程相关函数更加灵活，可以返回多个结果。

#### **22.** **Mysql 日志**

错误日志：记录启动，运行或者停止 mysql 时出现的问题； 通用日志：记录建立的客户端连接和执行的语句；

二进制日志：记录所有更改数据的语句；

203

慢查询日志：记录所有执行时间超过 long\_query\_time 秒的查询或者不适用索引的查询）

通过使用--slow\_query\_log[={0|1}]选项来启用慢查询日志，所有执行时间超多 long\_query\_time 的语句都会被记录。

## 二．MongoDB

#### **数据库的一些基本操作命令（列举一些常用命令即可）？**

MySQL 的常见命令如下：

* 1. create database name; 创建数据库
  2. use databasename; 选择数据库
  3. drop database name 直接删除数据库，不提醒
  4. show tables; 显示表
  5. describe tablename; 表的详细描述
  6. select 中加上 distinct 去除重复字段
  7. mysqladmin drop databasename 删除数据库前，有提示。
  8. 显示当前 mysql 版本和当前日期
  9. select version(),current\_date;

MongoDB 的常见命令如下：

1. db.help(); Help 查看命令提示
2. use yourDB; 切换/创建数据库
3. show dbs; 查询所有数据库
4. db.dropDatabase(); 删除当前使用数据库
5. db.getName(); 查看当前使用的数据库

204

1. db.version(); 当前 db 版本
2. db.addUser("name"); 添加用户
3. db.addUser("userName", "pwd123", true);
4. show users; 显示当前所有用户
5. db.removeUser("userName"); 删除用户11.db.yourColl.count(); 查询当前集合的数据条数

更多命令参考博客：<https://blog.csdn.net/wangpeng047/article/details/7705588>

|  |
| --- |
|  |
|  |  |

#### **Python 中调用mongo 数据库的包叫什么？**

Pymongo。

#### **3.** **MongoDB**

MongoDB 是一个面向文档的数据库系统。使用 C++编写，不支持 SQL，但有自己功能强大的查询语法。

MongoDB 使用 BSON 作为数据存储和传输的格式。BSON 是一种类似 JSON 的二进制序列化文档，支持嵌套对象和数组。

MongoDB 很像 MySQL，document 对应 MySQL 的 row，collection 对应 MySQL 的 table 应用场景：

* 1. 网站数据：mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。
  2. 缓存：由于性能很高，mongo 也适合作为信息基础设施的缓存层。在系统重启之后， 由 mongo 搭建的持久化缓存可以避免下层的数据源过载。
  3. 大尺寸、低价值的数据：使用传统的关系数据库存储一些数据时可能会比较贵，在此之前，很多程序员往往会选择传统的文件进行存储。
  4. 高伸缩性的场景：mongo 非常适合由数十或者数百台服务器组成的数据库。
  5. 用于对象及 JSON 数据的存储：mongo 的 BSON 数据格式非常适合文档格式化的存储及查询。
  6. 重要数据：mysql，一般数据：mongodb，临时数据：memcache
  7. 对于关系数据表而言，mongodb 是提供了一个更快速的视图 view；而对于 PHP 程序而言，mongodb 可以作为一个持久化的数组来使用，并且这个持久化的数组还可以支持排序、条件、限制等功能。
  8. 将mongodb 代替mysql 的部分功能，主要一个思考点就是：把 mongodb 当作mysql 的一个 view（视图），view 是将表数据整合成业务数据的关键。比如说对原始数据进行报表，那么就要先把原始数据统计后生成 view，在对 view 进行查询和报表。

不适合的场景：

1. 高度事物性的系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。
2. 传统的商业智能应用：针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。
3. 需要 SQL 的问题
4. 重要数据，关系数据

优点: 1）弱一致性（最终一致），更能保证用户的访问速度2）文档结构的存储方式，能够更便捷的获取数

* 1. 内置 GridFS，高效存储二进制大对象 (比如照片和视频)
  2. 支持复制集、主备、互为主备、自动分片等特性
  3. 动态查询
  4. 全索引支持,扩展到内部对象和内嵌数组缺点: 1）不支持事务

2）MongoDB 占用空间过大,维护工具不够成熟

#### **4.** **MongoDB 成为优秀的NoSQL 数据库的原因是什么?**

以下特点使得 MongoDB 成为优秀的 NoSQL 数据库： 1）面向文件的

1. 高性能
2. 高可用性
3. 易扩展性
4. 丰富的查询语言

207

#### **分析器在MongoDB 中的作用是什么?**

MongoDB 中包括了一个可以显示数据库中每个操作性能特点的数据库分析器。通过这个分析器你可以找到比预期慢的查询(或写操作);利用这一信息，比如，可以确定是否需要添加索引。

#### **怎么查看MongoDB 正在使用的链接?**

db.\_adminCommand("connPoolStats");

#### **MySQL 与MongoDB 本质之间最基本的差别是什么**

差别在多方面，例如：数据的表示、查询、关系、事务、模式的设计和定义、速度和性能。MongoDB 是由 C++语言编写的，是一个基于分布式文件存储的开源数据库系统。在高负载的

情况下，添加更多的节点，可以保证服务器性能。

MongoDB 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

MongoDB 是一个面向文档的数据库，目前由 10gen 开发并维护，它的功能丰富齐全，所以完全可以替代 MySQL。

与 MySQL 等关系型数据库相比，MongoDB 的优点如下：

①弱一致性，更能保证用户的访问速度。

②文档结构的存储方式，能够更便捷的获取数据。

③内置 GridFS，支持大容量的存储。

④内置 Sharding。

⑤第三方支持丰富。(这是与其他的 NoSQL 相比，MongoDB 也具有的优势)

⑥性能优越：

MongoDB 本身它还算比较年轻的一个产品，所以它的问题，就是成熟度肯定没有传统 MySQL 那么成熟稳定。所以在使用的时候，

第一，尽量使用稳定版，不要在线上使用开发版，这是一个大原则；

另外一点，备份很重要，MongoDB 如果出现一些异常情况，备份一定是要能跟上。除了通过传统的复制的方式来做备份，离线备份也还是要有，不管你是用什么方式，都要有一个完整的离线备份。往往最后出现了特殊情况，它能帮助到你；

另外，MongoDB 性能的一个关键点就是索引，索引是不是能有比较好的使用效率，索引是不是能够放在内存中，这样能够提升随机读写的性能。如果你的索引不能完全放在内存中，一旦出现随机读写比较高的时候，它就会频繁地进行磁盘交换，这个时候，MongoDB 的性能就会急剧下降，会出现波动。

另外，MongoDB 还有一个最大的缺点，就是它占用的空间很大，因为它属于典型空间换时间原则的类型。那么它的磁盘空间比普通数据库会浪费一些，而且到目前为止它还没有实现在线压缩功能， 在 MongoDB 中频繁的进行数据增删改时，如果记录变了，例如数据大小发生了变化，这时候容易产生一些数据碎片，出现碎片引发的结果，一个是索引会出现性能问题，

另外一个就是在一定的时间后，所占空间会莫明其妙地增大，所以要定期把数据库做修复，定期重新做索引，这样会提升 MongoDB 的稳定性和效率。在最新的版本里，它已经在实现在线压缩，估计应该在 2.0 版左右，应该能够实现在线压缩，可以在后台执行现在 repair DataBase 的一些操作。如果那样，就解决了目前困扰我们的大问题。

#### **使用MongoDB 的优点**

* 1. 面向文件

209

1. 高性能
2. 高可用
3. 易扩展
4. 可分片
5. 对数据存储友好

#### **关于MongoDB 更多知识请扫描二维码查看**

|  |
| --- |
|  |
|  |  |

## 三．Redis

#### **Redis 中list 底层实现有哪几种？有什么区别？**

列表对象的编码可以是 ziplist 或者 linkedlist

ziplist 是一种压缩链表，它的好处是更能节省内存空间，因为它所存储的内容都是在连续的内存区域当中的。当列表对象元素不大，每个元素也不大的时候，就采用 ziplist 存储。但当数据量过大时就 ziplist 就不是那么好用了。因为为了保证他存储内容在内存中的连续性，插入的复杂度是 O(N)，

即每次插入都会重新进行realloc。如下图所示，对象结构中ptr 所指向的就是一个ziplist。整个ziplist 只需要 malloc 一次，它们在内存中是一块连续的区域。

更多详细内容请见：<https://blog.csdn.net/caishenfans/article/details/44784131>

|  |
| --- |
|  |
|  |  |

#### **怎样解决数据库高并发的问题？**

解决数据库高并发的常见方案：

##### 缓存式的 Web 应用程序架构：

在 Web 层和 DB(数据库)层之间加一层 cache 层，主要目的：减少数据库读取负担，提高数据读取速度。cache 存取的媒介是内存，可以考虑采用分布式的 cache 层，这样更容易破除内存容量的限制，同时增加了灵活性。

##### 增加 Redis 缓存数据库：

参考博客：<https://www.cnblogs.com/Leo_wl/p/5791327.html>



##### 增加数据库索引

1. **页面静态化：**

效率最高、消耗最小的就是纯静态化的 html 页面，所以我们尽可能使我们的网站上的页面采用静态页面来实现，这个最简单的方法其实也是最有效的方法。用户可以直接获取页面，不用像 MVC 结构走那么多流程，比较适用于页面信息大量被前台程序调用，但是更新频率很小的情况。

##### 使用存储过程：

处理一次请求需要多次访问数据库的操作，可以把操作整合到储存过程，这样只要一次数据库访问就可以了。

##### MySQL 主从读写分离：

当数据库的写压力增加，cache 层（如 Memcached）只能缓解数据库的读取压力。读写集中在一个数据库上让数据库不堪重负。使用主从复制技术（master-slave 模式）来达到读写分离，以提高读写性能和读库的可扩展性。读写分离就是只在主服务器上写，只在从服务器上读，基本原理是让主数据库处理事务性查询，而从数据库处理 select 查询，数据库复制被用于把事务性查询（增删改） 导致的改变更新同步到集群中的从数据库。

MySQL 读写分离提升系统性能：

1、主从只负责各自的读和写，极大程度缓解 X 锁和 S 锁争用。

2、slave 可以配置 MyISAM 引擎，提升查询性能以及节约系统开销。

3、master 直接写是并发的，slave 通过主库发送来的 binlog 恢复数据是异步的。4、slave 可以单独设置一些参数来提升其读的性能。

5、增加冗余，提高可用性。

实现主从分离可以使用 MySQL 中间件如：Atlas

##### 分表分库：

在 cache 层的高速缓存，MySQL 的主从复制，读写分离的基础上，这时 MySQL 主库的写压力开始出现瓶颈，而数据量的持续猛增，由于 MyISAM 使用表锁，在高并发下会出现严重的锁问题，大量的高并发 MySQL 应用开始使用 InnoDB 引擎代替 MyISAM。采用 Master-Slave 复制模式的 MySQL 架构，只能对数据库的读进行扩展，而对数据的写操作还是集中在 Master 上。这时需要对数据库的吞吐能力进一步地扩展，以满足高并发访问与海量数据存储的需求。

对于访问极为频繁且数据量巨大的单表来说，首先要做的是减少单表的记录条数，以便减少数据查询所需的时间，提高数据库的吞吐，这就是所谓的**分表【水平拆分】**。在分表之前，首先需要选择适当的分表策略（尽量避免分出来的多表关联查询），使得数据能够较为均衡地分布到多张表中， 并且不影响正常的查询。

分表能够解决单表数据量过大带来的查询效率下降的问题，但是却无法给数据库的并发处理能力带来质的提升。面对高并发的读写访问，当数据库 master 服务器无法承载写操作压力时，不管如何扩展 Slave 服务器都是没有意义的，对数据库进行拆分，从而提高数据库写入能力，即**分库【垂直拆分】**。

##### 负载均衡集群：

将大量的并发请求分担到多个处理节点。由于单个处理节点的故障不影响整个服务，负载均衡集群同时也实现了高可用性。

负载均衡将是大型网站解决高负荷访问和大量并发请求采用的终极解决办法。

#### **3.** **Redis 集群实现？**

需要结合真实案例去分析，这里给大家推荐一个不错的博客。<https://blog.csdn.net/yfkiss/article/details/38944179>

|  |
| --- |
|  |
|  |  |

#### **Redis 数据库，内容是以何种结构存放在Redis 中的？**

String（字符串），Hash（哈希），List（列表），Set（集合）及 zset(sortedset：有序集合)。

#### **Redis 的并发竞争问题怎么解决？**

方案一：可以使用独占锁的方式，类似操作系统的 mutex 机制，不过实现相对复杂，成本较高。<https://blog.csdn.net/black_ox/article/details/48972085>



方案二：使用乐观锁的方式进行解决（成本较低，非阻塞，性能较高） 如何用乐观锁方式进行解决？

本质上是假设不会进行冲突，使用 redis 的命令 watch 进行构造条件

#### **MySQL 和Redis 高可用性体现在哪些方面？**

1. MySQL Replication 是 MySQL 官方提供的主从同步方案，用于将一个 MySQL 实例的数据，同步到另一个实例中。Replication 为保证 数据安全做了重要的保证，也是现在运用最广的 MySQL 容灾方案。Replication 用两个或以上的实例搭建了 MySQL 主从复制集群， 提供 单点写入，多点读取的服务，实现了读的 scale out。
2. Sentinel 是 Redis 官方为集群提供的高可用解决方案。在实际 项目中可以使用 sentinel 去做 Redis 自动故障转移，减少人工介入的工作量。另外 sentinel 也给客户端提供了监控消息的通知，这样客户端就可根据消息类型去判断服务器的状态，去做对应的适配操作。
3. 下面是 Sentinel 主要功能列表：

Monitoring：Sentinel 持续检查集群中的 master、slave 状态，判断是否存活。

Notification：在发现某个 Redis 实例死的情况下，Sentinel 能通过 API 通知系统管理员或其他程序脚本。

Automatic failover：如果一个 master 挂掉后，sentinel 立马启动故障转移，把某个 slave 提升为 master。其他的 slave 重新配置指向新 master。

Configuration provider：对于客户端来说 sentinel 通知是有效可信赖的。客户端会连接sentinel 去请求当前 master 的地址，一旦发生故障 sentinel 会提供新地址给客户端。

#### **Redis 和MongoDB 的优缺点**

MongoDB 和 Redis 都是 NoSQL，采用结构型数据存储。二者在使用场景中，存在一定的区别， 这也主要由于二者在内存映射的处理过程，持久化的处理方法不同。MongoDB 建议集群部署，更多的考虑到集群方案，Redis 更偏重于进程顺序写入，虽然支持集群，也仅限于主-从模式。

Redis 优点：

1. 读写性能优异
2. 支持数据持久化，支持 AOF 和 RDB 两种持久化方式
3. 支持主从复制，主机会自动将数据同步到从机，可以进行读写分离。
4. 数据结构丰富：数据结构丰富：支持 string、hash、set、sortedset、list 等数据结构。缺点：
5. Redis 不具备自动容错和恢复功能，主机从机的宕机都会导致前端部分读写请求失败，需要等待机器重启或者手动切换前端的 IP 才能恢复。
6. 主机宕机，宕机前有部分数据未能及时同步到从机，切换 IP 后还会引入数据不一致的问题， 降低了系统的可用性。

1. Redis 的主从复制采用全量复制，复制过程中主机会 fork 出一个子进程对内存做一份快照， 并将子进程的内存快照保存为文件发送给从机，这一过程需要确保主机有足够多的空余内 存。若快照文件较大，对集群的服务能力会产生较大的影响，而且复制过程是在从机新加入集群或者从机和主机网络断开重连时都会进行，也就是网络波动都会造成主机和从机间的一次全量的数据复制，这对实际的系统运营造成了不小的麻烦。
2. Redis 较难支持在线扩容，在集群容量达到上限时在线扩容会变得很复杂。为避免这一问题， 运维人员在系统上线时必须确保有足够的空间，这对资源造成了很大的浪费。

MongoDB 优点:

a）弱一致性（最终一致），更能保证用户的访问速度b）文档结构的存储方式，能够更便捷的获取数

c）内置 GridFS，高效存储二进制大对象 (比如照片和视频) d）支持复制集、主备、互为主备、自动分片等特性

1. 动态查询
2. 全索引支持,扩展到内部对象和内嵌数组缺点：

a）不支持事务b）MongoDB 占用空间过大c）维护工具不够成熟

#### **Redis 基本类型、相关方法**

Redis 支持五种数据类型：string（字符串）、hash（哈希）、list（列表）、set（集合）及 zset(sorted set：有序集合)。

217

一 、 String

1. String 是 Redis 最为常用的一种数据类型，String 的数据结构为 key/value 类型，String 可以包含任何数据。
2. 常用命令: set,get,decr,incr,mget 等。二 、 Hash

1.Hash 类型可以看成是一个 key/value 都是 String 的 Map 容器。2.常用命令：hget,hset,hgetall 等。

三 、 List

1. List 用于存储一个有序的字符串列表，常用的操作是向队列两 端添加元素或者获得列表的某一片段。
2. 常用命令：lpush,rpush,lpop,rpop,lrange 等四 、 Set
3. Set 可以理解为一组无序的字符集合，Set 中相同的元素是不会重复出现的，相同的元素只保留一个。
4. 常用命令：sadd,spop,smembers,sunion 等。五、 Sorted Set（有序集合）
5. 有序集合是在集合的基础上为每一个元素关联一个分数，Redis 通过分数为集合中的成员进行排序。
6. 常用命令：zadd,zrange,zrem,zcard 等。

#### **9.** **Redis 的事务？**

一、 Redis 事务允许一组命令在单一步骤中执行。事务有两个属性，说明如下：

* 1. 事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。
  2. Redis 事务是原子的。原子意味着要么所有的命令都执行，要么都不执行； 二、 一个事务从开始到执行会经历以下三个阶段：

a)开始事务b)命令入队c)执行事务

#### **10.** **Redis 的使用场景有哪些？**

1. 取最新 N 个数据的操作
2. 排行榜应用,取 TOP N 操作
3. 需要精准设定过期时间的应用
4. 计数器应用
5. uniq 操作,获取某段时间所有数据排重值
6. Pub/Sub 构建实时消息系统
7. 构建队列系统
8. 缓存

#### **11.** **Redis 默认端口，默认过期时间，Value 最多可以容纳的数据 长度？**

1.默认端口：6379

1. 默认过期时间：可以说永不过期，一般情况下，当配置中开启 了超出最大内存限制就写磁盘的话， 那么没有设置过期时间的 key 可能会被写到磁盘上。假如没设置，那么 REDIS 将使用 LRU 机制，将 内存中的老数据删除，并写入新数据。
2. Value 最多可以容纳的数据长度是：512M。

#### **12.** **Redis 有多少个库？**

Redis 一个实例下有 16 个。