算法面试必备-----数据分析常见面试题

算法面试必备-----数据分析常见面试题

1、统计学问题

问题：贝叶斯公式复述并解释应用场景

问题：朴素贝叶斯的理解

问题：参数估计

问题：极大似然估计

问题：假设检验

问题：P值是什么？

问题：置信度、置信区间

问题：协方差与相关系数的区别和联系

问题： 中心极限定理

2、概率问题

问题：扑克牌54张，分成2份，求这2份都有2张A的概率

问题：男生点击率增加，女生点击率增加，总体为何减少?

3、数据库问题

问题：什么是数据库,数据库管理系统,数据库系统,数据库管理员?

问题：什么是元组,码,候选码,主码,外码,主属性,非主属性？

问题：主键和外键有什么区别?

问题：数据库范式了解吗?

问题：什么是存储过程?

问题：drop、delete与truncate区别？

问题：DML 语句和 DDL 语句区别：

问题：数据库设计通常分为哪几步?

问题：事务的ACID特性是什么？

问题：并发事务带来哪些问题?

延伸问题：不可重复度和幻读有什么区别？

问题：事务隔离级别有哪些? MySQL的默认隔离级别是?

问题：乐观锁与悲观锁的区别

问题：乐观锁常见的两种实现方式

问题：乐观锁的缺点

问题：数据库的索引

1、什么是数据库的索引

2、数据库的索引的优缺点

3、一般来说，应该在这些列上创建索引

4、一般来说，不应该创建索引的的这些列具有下列特点

5、数据库设计器中创建三种索引：唯一索引、主键索引和聚集索引

6、MyISAM 和 InnoDB 两个存储引擎的索引

\*\*MyISAM 索引实现\*\*

InnoDB 索引实现

问题：B树和B+树有什么区别？

延伸问题：与哈希索引的区别？

问题：MyISAM 和 InnoDB 的区别有哪些？

问题：延伸问题：如何选择存储引擎？

问题：MySQL主从复制是怎么做的？

问题：大表优化

问题：如何保证缓存与数据库双写时的数据一致性?

问题：红黑树和AVL树有什么区别？

问题：Redis 和 Memcached 的区别

问题：什么是AOF重写？

问题：数据库与数据仓库的区别

问题：SQL的数据类型

问题：left join,right join,inner join,full join之间的区别？

问题：having和where的区别？

问题：not in和not exists区别

问题：mysql中设置row number

问题：sql中null与‘ ’的区别。

问题：mysql 中视图和表的区别以及联系是什么？

问题：行存储和列存储的区别。

问题：如何写SQL求出中位数平均数和众数（除了用count之外的方法）

4、机器学习问题

问题：如何避免决策树过拟合？

问题：请说明随机森林较一般决策树稳定的几点原因

问题：SVM的优点

问题：K-means的原理

问题：K-Means算法原理及改进，遇到异常值怎么办？评估算法的指标有哪些？

问题：PCA（主成分分析）

问题：时间序列分析

问题：数据预处理过程有哪些？

问题：数据清理中，处理缺失值的方法是？

问题： 余弦距离与欧式距离求相似度的差别？

问题：分类算法性能的主要评价指标

问题： GBDT（梯度提升树）

5、业务场景问题

问题：处理需求时的一般思路是什么？，并举例

问题：业务场景题，如何分析次日留存率下降的问题？

问题：给你一个无序数组，怎么才能合理采样？

问题：如果次日用户留存率下降了 5%该怎么分析？

问题： 卖玉米如何提高收益？价格提高多少才能获取最大收益？

问题：怎么做恶意刷单检测?

问题：类比到头条的收益，头条放多少广告可以获得最大收益，不需要真的计算，只要有个思路就行？

费米估计

问题：不用任何公开参考资料，估算今年新生儿出生数量

AB测试

问题：A/B test是什么？

问题：A/B test工作原理

问题：进行A/B test的目的是什么？

问题：A/B test流程

问题：A/B test需要注意的点

问题：A/B test中要知道的统计学知识

问题：A/B test简例（结合Python实现）

数据埋点

问题：什么是埋点？

问题：埋点方式有哪些？

如何评估一个活动

问题：评估一个活动的思路

6、Python问题

问题：常用的Python库有哪些？

7、大数据问题：

问题：hadoop原理和mapreduce原理

手写SQL

问题：有一张用户签到表【t\_user\_attendence】，标记每天用户是否签到（说明：该表包含所有用户所有工作日的出勤记录） ，包含三个字段：日期【fdate】，用户id【fuser\_id】，用户当天是否签到【fis\_sign\_in：0否1是】；

问题1：请计算截至当前每个用户已经连续签到的天数（输出表仅包含当天签到的所有用户，计算其连续签到天数）

问题2：请计算每个用户历史以来最大的连续签到天数（输出表为用户签到表中所有出现过的用户，计算其历史最大连续签到天数）

手写Python

题目：针对股票的最大回撤率指标定义，给出代码实现思路。给定的是产品所有交易日的净值序列，且其净值序列已按照日期排序。

算法面试必备-----数据分析常见面试题

数据分析流程：

1、统计学问题

问题：贝叶斯公式复述并解释应用场景

1）P（A|B) = P(B|A)\*P(A) / P(B)

2）如搜索query纠错，设A为正确的词，B为输入的词，那么：

a. P(A|B)表示输入词B实际为A的概率

b. P(B|A)表示词A错输为B的概率，可以根据AB的相似度计算（如编辑距离）

c. P(A)是词A出现的频率，统计获得

d. P(B)对于所有候选的A都一样，所以可以省去

1

2

3

4

5

6

7

问题：朴素贝叶斯的理解

1）推导（参考）

2）理解：朴素贝叶斯是在已知一些先验概率的情况下，由果索因的一种方法

3）其它：朴素的意思是假设了事件相互独立

问题：参数估计

用样本统计量去估计总体的参数。’

参考

问题：极大似然估计

利用已知的样本结果，反推最有可能（最大概率）导致这样结果的参数值。

参考

问题：假设检验

参数估计和假设检验是统计推断的两个组成部分，它们都是利用样本对总体进行某种推断，但推断的角度不同。

参数估计讨论的是用样本估计总体参数的方法，总体参数μ在估计前是未知的。

而在假设检验中，则是先对μ的值提出一个假设，然后利用样本信息去检验这个假设是否成立。

参考

问题：P值是什么？

P值是用来判定假设检验结果的一个参数，也可以根据不同的分布使用分布的拒绝域进行比较。由R·A·Fisher首先提出。

P值（P value）就是当原假设为真时所得到的样本观察结果或更极端结果出现的概率。如果P值很小，说明原假设情况的发生的概率很小，而如果出现了，根据小概率原理，我们就有理由拒绝原假设，P值越小，我们拒绝原假设的理由越充分。总之，P值越小，表明结果越显著。但是检验的结果究竟是“显著的”、“中度显著的”还是“高度显著的”需要我们自己根据P值的大小和实际问题来解决。

问题：置信度、置信区间

置信区间是我们所计算出的变量存在的范围

置信水平就是我们对于这个数值存在于我们计算出的这个范围的可信程度。

举例来讲，有95%的把握，真正的数值在我们所计算的范围里。

在这里，95%是置信水平，而计算出的范围，就是置信区间。

如果置信度为95%， 则抽取100个样本来估计总体的均值，由100个样本所构造的100个区间中，约有95个区间包含总体均值。

问题：协方差与相关系数的区别和联系

协方差：

协方差表示的是两个变量的总体的误差，这与只表示一个变量误差的方差不同。 如果两个变量的变化趋势一致，也就是说如果其中一个大于自身的期望值，另外一个也大于自身的期望值，那么两个变量之间的协方差就是正值。 如果两个变量的变化趋势相反，即其中一个大于自身的期望值，另外一个却小于自身的期望值，那么两个变量之间的协方差就是负值。

相关系数：

研究变量之间线性相关程度的量，取值范围是[-1,1]。相关系数也可以看成协方差：一种剔除了两个变量量纲影响、标准化后的特殊协方差。

问题： 中心极限定理

中心极限定理定义：

（1）任何一个样本的平均值将会约等于其所在总体的平均值。

（2）不管总体是什么分布，任意一个总体的样本平均值都会围绕在总体的平均值周围，并且呈正态分布。

中心极限定理作用：

（1）在没有办法得到总体全部数据的情况下，我们可以用样本来估计总体。

（2）根据总体的平均值和标准差，判断某个样本是否属于总体。

2、概率问题

问题：扑克牌54张，分成2份，求这2份都有2张A的概率

M表示两个牌堆各有2个A的情况：M=4(25!25!)

N表示两个牌堆完全随机的情况：N=27!27!

所以概率为：M/N = 926/53\*17

问题：男生点击率增加，女生点击率增加，总体为何减少?

因为男女的点击率可能有较大差异，同时低点击率群体的占比增大。

如原来男性20人，点击1人；女性100人，点击99人，总点击率100/120。

现在男性100人，点击6人；女性20人，点击20人，总点击率26/120。

即那个段子“A系中智商最低的人去读B，同时提高了A系和B系的平均智商。”

3、数据库问题

问题：什么是数据库,数据库管理系统,数据库系统,数据库管理员?

数据库 :数据库(DataBase简称DB)就是信息的集合或者说数据库是由数据库管理系统管理的数据的集合。

数据库管理系统 : 数据库管理系统(Database Management System 简称DBMS)是一种操纵和管理数据库的大型软件，通常用语用于建立、使用和维护数据库。

数据库系统 : 数据库系统(Data Base System，简称DBS)通常由软件、数据库和数据管理员(DBA)组成。

数据库管理员 : 数据库管理员(Database Administrator,简称DBA)负责全面管理和控制数据库系统。

问题：什么是元组,码,候选码,主码,外码,主属性,非主属性？

元组 ： 元组（tuple）是关系数据库中的基本概念，关系是一张表，表中的每行（即数据库中的每条记录）就是一个元组，每列就是一个属性。 在二维表里，元组也称为行。

码 ：码就是能唯一标识实体的属性，对应表中的列。

候选码 ： 若关系中的某一属性或属性组的值能唯一的标识一个元组，而其任何、子集都不能再标识，则称该属性组为候选码。例如：在学生实体中，“学号”是能唯一的区分学生实体的，同时又假设“姓名”、“班级”的属性组合足以区分学生实体，那么{学号}和{姓名，班级}都是候选码。

主码 : 主码也叫主键。主码是从候选码中选出来的。 一个实体集中只能有一个主码，但可以有多个候选码。

外码 : 外码也叫外键。如果一个关系中的一个属性是另外一个关系中的主码则这个属性为外码。

主属性 ： 候选码中出现过的属性称为主属性。比如关系 工人（工号，身份证号，姓名，性别，部门）.显然工号和身份证号都能够唯一标示这个关系，所以都是候选码。工号、身份证号这两个属性就是主属性。如果主码是一个属性组，那么属性组中的属性都是主属性。

非主属性： 不包含在任何一个候选码中的属性称为非主属性。比如在关系——学生（学号，姓名，年龄，性别，班级）中，主码是“学号”，那么其他的“姓名”、“年龄”、“性别”、“班级”就都可以称为非主属性。

问题：主键和外键有什么区别?

主键(主码) ：主键用于唯一标识一个元组，不能有重复，不允许为空。一个表只能有一个主键。

外键(外码) ：外键用来和其他表建立联系用，外键是另一表的主键，外键是可以有重复的，可以是空值。一个表可以有多个外键。

问题：数据库范式了解吗?

1NF(第一范式)

属性（对应于表中的字段）不能再被分割，也就是这个字段只能是一个值，不能再分为多个其他的字段了。1NF是所有关系型数据库的最基本要求 ，也就是说关系型数据库中创建的表一定满足第一范式。

2NF(第二范式)

2NF在1NF的基础之上，消除了非主属性对于码的部分函数依赖。如下图所示，展示了第一范式到第二范式的过渡。第二范式在第一范式的基础上增加了一个列，这个列称为主键，非主属性都依赖于主键。

3NF(第三范式)

3NF在2NF的基础之上，消除了非主属性对于码的传递函数依赖 。符合3NF要求的数据库设计，基本上解决了数据冗余过大，插入异常，修改异常，删除异常的问题。比如在关系R(学号 ,姓名, 系名，系主任)中，学号 → 系名，系名 → 系主任，所以存在非主属性系主任对于学号的传递函数依赖，所以该表的设计，不符合3NF的要求。

总结

1NF：属性不可再分。

2NF：1NF的基础之上，消除了非主属性对于码的部分函数依赖。

3NF：3NF在2NF的基础之上，消除了非主属性对于码的传递函数依赖 。

一些重要的概念：

函数依赖（functional dependency） ：若在一张表中，在属性（或属性组）X的值确定的情况下，必定能确定属性Y的值，那么就可以说Y函数依赖于X，写作 X → Y。

部分函数依赖（partial functional dependency） ：如果X→Y，并且存在X的一个真子集X0，使得X0→Y，则称Y对X部分函数依赖。比如学生基本信息表R中（学号，身份证号，姓名）当然学号属性取值是唯一的，在R关系中，（学号，身份证号）->（姓名），（学号）->（姓名），（身份证号）->（姓名）；所以姓名部分函数依赖与（学号，身份证号）；

完全函数依赖(Full functional dependency) ：在一个关系中，若某个非主属性数据项依赖于全部关键字称之为完全函数依赖。比如学生基本信息表R（学号，班级，姓名）假设不同的班级学号有相同的，班级内学号不能相同，在R关系中，（学号，班级）->（姓名），但是（学号）->(姓名)不成立，（班级）->(姓名)不成立，所以姓名完全函数依赖与（学号，班级）；

传递函数依赖 ： 在关系模式R(U)中，设X，Y，Z是U的不同的属性子集，如果X确定Y、Y确定Z，且有X不包含Y，Y不确定X，（X∪Y）∩Z=空集合，则称Z传递函数依赖(transitive functional dependency) 于X。传递函数依赖会导致数据冗余和异常。传递函数依赖的Y和Z子集往往同属于某一个事物，因此可将其合并放到一个表中。比如在关系R(学号 ,姓名, 系名，系主任)中，学号 → 系名，系名 → 系主任，所以存在非主属性系主任对于学号的传递函数依赖。。

问题：什么是存储过程?

我们可以把存储过程看成是一些 SQL 语句的集合，中间加了点逻辑控制语句。存储过程在业务比较复杂的时候是非常实用的，比如很多时候我们完成一个操作可能需要写一大串SQL语句，这时候我们就可以写有一个存储过程，这样也方便了我们下一次的调用。存储过程一旦调试完成通过后就能稳定运行，另外，使用存储过程比单纯SQL语句执行要快，因为存储过程是预编译过的。

存储过程在互联网公司应用不多，因为存储过程难以调试和扩展，而且没有移植性，还会消耗数据库资源。

问题：drop、delete与truncate区别？

用法不同

drop(丢弃数据): drop table 表名 ，直接将表都删除掉，在删除表的时候使用。

truncate (清空数据) : truncate table 表名 ，只删除表中的数据，再插入数据的时候自增长id又从1开始，在清空表中数据的时候使用。

delete（删除数据） : delete from 表名 where 列名=值，删除某一列的数据，如果不加 where 子句和truncate table 表名作用类似。

truncate 和不带 where 子句的 delete、以及 drop 都会删除表内的数据，但是 truncate 和 delete 只删除数据不删除表的结构(定义)，执行drop语句，此表的结构也会删除，也就是执行 drop 之后对应的表不复存在。

属于不同的数据库语言

truncate和drop 属于DDL(数据定义语言)语句，操作立即生效，原数据不放到 rollback segment 中，不能回滚，操作不触发 trigger。而 delete 语句是DML (数据库操作语言)语句，这个操作会放到 rollback segement 中，事务提交之后才生效。

执行速度不同

一般来说:drop>truncate>delete。

问题：DML 语句和 DDL 语句区别：

DML 是数据库操作语言（Data Manipulation Language）的缩写，是指对数据库中表记录的操作，主要包括表记录的插入（insert）、更新（update）、删除（delete）和查询（select），是开发人员日常使用最频繁的操作。

DDL （Data Definition Language）是数据定义语言的缩写，简单来说，就是对数据库内部的对象进行创建、删除、修改的操作语言。它和 DML 语言的最大区别是 DML 只是对表内部数据的操作，而不涉及到表的定义、结构的修改，更不会涉及到其他对象。DDL 语句更多的被数据库管理员（DBA）所使用，一般的开发人员很少使用。

问题：数据库设计通常分为哪几步?

需求分析 : 分析用户的需求，包括数据、功能和性能需求。

概念结构设计 : 主要采用E-R模型进行设计，包括画E-R图。

逻辑结构设计 : 通过将E-R图转换成表，实现从E-R模型到关系模型的转换。

物理结构设计 : 主要是为所设计的数据库选择合适的存储结构和存取路径。

数据库实施 : 包括编程、测试和试运行

数据库的运行和维护 : 系统的运行与数据库的日常维护。

问题：事务的ACID特性是什么？

原子性： 事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；

一致性： 执行事务前后，数据保持一致，多个事务对同一个数据读取的结果是相同的；

隔离性： 并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；

持久性： 一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。

问题：并发事务带来哪些问题?

在典型的应用程序中，多个事务并发运行，经常会操作相同的数据来完成各自的任务（多个用户对统一数据进行操作）。并发虽然是必须的，但可能会导致以下的问题。

脏读（Dirty read）: 当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。

丢失修改（Lost to modify）: 指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修改。 例如：事务1读取某表中的数据A=20，事务2也读取A=20，事务1修改A=A-1，事务2也修改A=A-1，最终结果A=19，事务1的修改被丢失。

不可重复读（Unrepeatableread）: 指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。

幻读（Phantom read）: 幻读与不可重复读类似。它发生在一个事务（T1）读取了几行数据，接着另一个并发事务（T2）插入了一些数据时。在随后的查询中，第一个事务（T1）就会发现多了一些原本不存在的记录，就好像发生了幻觉一样，所以称为幻读。

延伸问题：不可重复度和幻读有什么区别？

不可重复读的重点是修改，幻读的重点在于新增或者删除。

问题：事务隔离级别有哪些? MySQL的默认隔离级别是?

问题：乐观锁与悲观锁的区别

悲观锁

总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁（共享资源每次只给一个线程使用，其它线程阻塞，用完后再把资源转让给其它线程）。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。Java中synchronized和ReentrantLock等独占锁就是悲观锁思想的实现。

乐观锁

总是假设最好的情况，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和CAS算法实现。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库提供的类似于write\_condition机制，其实都是提供的乐观锁。在Java中java.util.concurrent.atomic包下面的原子变量类就是使用了乐观锁的一种实现方式CAS实现的。

两种锁的使用场景

从上面对两种锁的介绍，我们知道两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下（多读场景），即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果是多写的情况，一般会经常产生冲突，这就会导致上层应用会不断的进行retry，这样反倒是降低了性能，所以一般多写的场景下用悲观锁就比较合适。

问题：乐观锁常见的两种实现方式

版本号机制

一般是在数据表中加上一个数据版本号version字段，表示数据被修改的次数，当数据被修改时，version值会加一。当线程A要更新数据值时，在读取数据的同时也会读取version值，在提交更新时，若刚才读取到的version值为当前数据库中的version值相等时才更新，否则重试更新操作，直到更新成功。

举一个简单的例子： 假设数据库中帐户信息表中有一个 version 字段，当前值为 1 ；而当前帐户余额字段（ balance ）为 $100 。

操作员 A 此时将其读出（ version=1 ），并从其帐户余额中扣除 （50（100-$50 ）。

在操作员 A 操作的过程中，操作员B 也读入此用户信息（ version=1 ），并从其帐户余额中扣除 （20（100-$20 ）。

操作员 A 完成了修改工作，将数据版本号加一（ version=2 ），连同帐户扣除后余额（ balance=$50 ），提交至数据库更新，此时由于提交数据版本大于数据库记录当前版本，数据被更新，数据库记录 version 更新为 2 。

操作员 B 完成了操作，也将版本号加一（ version=2 ）试图向数据库提交数据（ balance=$80 ），但此时比对数据库记录版本时发现，操作员 B 提交的数据版本号为 2 ，数据库记录当前版本也为 2 ，不满足 “ 提交版本必须大于记录当前版本才能执行更新 “ 的乐观锁策略，因此，操作员 B 的提交被驳回。

这样，就避免了操作员 B 用基于 version=1 的旧数据修改的结果覆盖操作员A 的操作结果的可能。

CAS算法

即compare and swap（比较与交换），是一种有名的无锁算法。无锁编程，即不使用锁的情况下实现多线程之间的变量同步，也就是在没有线程被阻塞的情况下实现变量的同步，所以也叫非阻塞同步（Non-blocking Synchronization）。CAS算法涉及到三个操作数

需要读写的内存值 V

进行比较的值 A

拟写入的新值 B

当且仅当 V 的值等于 A时，CAS通过原子方式用新值B来更新V的值，否则不会执行任何操作（比较和替换是一个原子操作）。一般情况下是一个自旋操作，即不断的重试。

问题：乐观锁的缺点

1 ABA 问题

如果一个变量V初次读取的时候是A值，并且在准备赋值的时候检查到它仍然是A值，那我们就能说明它的值没有被其他线程修改过了吗？很明显是不能的，因为在这段时间它的值可能被改为其他值，然后又改回A，那CAS操作就会误认为它从来没有被修改过。这个问题被称为CAS操作的 "ABA"问题。

JDK 1.5 以后的 AtomicStampedReference 类就提供了此种能力，其中的 compareAndSet 方法就是首先检查当前引用是否等于预期引用，并且当前标志是否等于预期标志，如果全部相等，则以原子方式将该引用和该标志的值设置为给定的更新值。

2 循环时间长开销大

自旋CAS（也就是不成功就一直循环执行直到成功）如果长时间不成功，会给CPU带来非常大的执行开销。 如果JVM能支持处理器提供的pause指令那么效率会有一定的提升，pause指令有两个作用，第一它可以延迟流水线执行指令（de-pipeline）,使CPU不会消耗过多的执行资源，延迟的时间取决于具体实现的版本，在一些处理器上延迟时间是零。第二它可以避免在退出循环的时候因内存顺序冲突（memory order violation）而引起CPU流水线被清空（CPU pipeline flush），从而提高CPU的执行效率。

3 只能保证一个共享变量的原子操作

CAS 只对单个共享变量有效，当操作涉及跨多个共享变量时 CAS 无效。但是从 JDK 1.5开始，提供了AtomicReference类来保证引用对象之间的原子性，你可以把多个变量放在一个对象里来进行 CAS 操作.所以我们可以使用锁或者利用AtomicReference类把多个共享变量合并成一个共享变量来操作。

问题：数据库的索引

1、什么是数据库的索引

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。

实现：索引的实现通常使用 B 树及其变种 B+ 树。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。

上图显示了一种索引方式。左边是数据库中的数据表，有col1和col2两个字段，一共有15条记录；右边是以col2列为索引列的B\_TREE索引，每个节点包含索引的键值和对应数据表地址的指针，这样就可以都过B\_TREE在O(logn)的时间复杂度内获取相应的数据，这样明显地加快了检索的速度。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

2、数据库的索引的优缺点

优点：创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

缺点：

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

3、一般来说，应该在这些列上创建索引

在经常需要搜索的列上，可以加快搜索的速度；在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

在经常使用在 WHERE 子句中的列上面创建索引，加快条件的判断速度。

4、一般来说，不应该创建索引的的这些列具有下列特点

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

5、数据库设计器中创建三种索引：唯一索引、主键索引和聚集索引

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在 employee 表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。

主键索引数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

聚集索引在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

6、MyISAM 和 InnoDB 两个存储引擎的索引

MyISAM 索引实现

MyISAM 引擎使用 B+Tree 作为索引结构,叶节点的 data 域存放的是数据记录的地址。下图是 MyISAM 索引的原理图:

这里设表一共有三列,假设我们以 Col1 为主键,则图 8 是一个 MyISAM 表的主索引(Primary key)示意。可以看出 MyISAM 的索引文件仅仅保存数据记录的地址。

辅助索引

在 MyISAM 中,主索引和辅助索引(Secondary key)在结构上没有任何区别,只是主索引要求 key 是唯一的,而辅助索引的 key 可以重复。如果我们在 Col2 上建立一个辅助索引,则此索引的结构如下图所示

同样也是一颗 B+Tree,data 域保存数据记录的地址。因此,MyISAM 中索引检索的算法为首先按照 B+Tree 搜索算法搜索索引,如果指定的 Key 存在,则取出其data 域的值,然后以 data 域的值为地址,读取相应数据记录。

MyISAM 的索引方式也叫做“非聚集索引”,之所以这么称呼是为了与 InnoDB的聚集索引区分。

InnoDB 索引实现

虽然 InnoDB 也使用 B+Tree 作为索引结构,但具体实现方式却与 MyISAM 截然不同。

1.第一个重大区别是 InnoDB 的数据文件本身就是索引文件。从上文知道,MyISAM 索引文件和数据文件是分离的,索引文件仅保存数据记录的地址。

而在InnoDB 中,表数据文件本身就是按 B+Tree 组织的一个索引结构,这棵树的叶点data 域保存了完整的数据记录。这个索引的 key 是数据表的主键,因此 InnoDB 表数据文件本身就是主索引。

上图是 InnoDB 主索引(同时也是数据文件)的示意图,可以看到叶节点包含了完整的数据记录。这种索引叫做聚集索引。因为 InnoDB 的数据文件本身要按主键聚集,

1 .InnoDB 要求表必须有主键(MyISAM 可以没有),如果没有显式指定,则 MySQL系统会自动选择一个可以唯一标识数据记录的列作为主键,如果不存在这种列,则MySQL 自动为 InnoDB 表生成一个隐含字段作为主键,类型为长整形。

同时,请尽量在 InnoDB 上采用自增字段做表的主键。因为 InnoDB 数据文件本身是一棵B+Tree,非单调的主键会造成在插入新记录时数据文件为了维持 B+Tree 的特性而频繁的分裂调整,十分低效,而使用自增字段作为主键则是一个很好的选择。如果表使用自增主键,那么每次插入新的记录,记录就会顺序添加到当前索引节点的后续位置,当一页写满,就会自动开辟一个新的页。如下图所示:

这样就会形成一个紧凑的索引结构,近似顺序填满。由于每次插入时也不需要移动已有数据,因此效率很高,也不会增加很多开销在维护索引上。

2.第二个与 MyISAM 索引的不同是 InnoDB 的辅助索引 data 域存储相应记录主键的值而不是地址。换句话说,InnoDB 的所有辅助索引都引用主键作为 data 域。

例如,图 11 为定义在 Col3 上的一个辅助索引:

问题：B树和B+树有什么区别？

B树是一颗多路平衡查找树。

每个节点最多有m-1个关键字（可以存有的键值对）。

根节点最少可以只有1个关键字。

非根节点至少有m/2个关键字。

每个节点中的关键字都按照从小到大的顺序排列，每个关键字的左子树中的所有关键字都小于它，而右子树中的所有关键字都大于它。

所有叶子节点都位于同一层，或者说根节点到每个叶子节点的长度都相同。

每个节点都存有索引和数据，也就是对应的key和value。

所以，根节点的关键字数量范围：1 <= k <= m-1，非根节点的关键字数量范围：m/2 <= k <= m-1。

另外，我们需要注意一个概念，描述一颗B树时需要指定它的阶数，阶数表示了一个节点最多有多少个孩子节点，一般用字母m表示阶数。

B+树其实和B树是非常相似的，我们首先看看相同点。

根节点至少一个元素

非根节点元素范围：m/2 <= k <= m-1

不同点。

B+树有两种类型的节点：内部结点（也称索引结点）和叶子结点。内部节点就是非叶子节点，内部节点不存储数据，只存储索引，数据都存储在叶子节点。

内部结点中的key都按照从小到大的顺序排列，对于内部结点中的一个key，左树中的所有key都小于它，右子树中的key都大于等于它。叶子结点中的记录也按照key的大小排列。

每个叶子结点都存有相邻叶子结点的指针，叶子结点本身依关键字的大小自小而大顺序链接。

父节点存有右孩子的第一个元素的索引。

B+树相对于B树有一些自己的优势，可以归结为下面几点。

单一节点存储的元素更多，使得查询的IO次数更少，所以也就使得它更适合做为数据库MySQL的底层数据结构了。

所有的查询都要查找到叶子节点，查询性能是稳定的，而B树，每个节点都可以查找到数据，所以不稳定。

所有的叶子节点形成了一个有序链表，更加便于查找。

参考：

B+树的非叶子节点只是存储key，占用空间非常小，因此每一层的节点能索引到的数据范围更加的广。换句话说，每次IO操作可以搜索更多的数据。

叶子节点两两相连，符合磁盘的预读特性。比如叶子节点存储50和55，它有个指针指向了60和62这个叶子节点，那么当我们从磁盘读取50和55对应的数据的时候，由于磁盘的预读特性，会顺便把60和62对应的数据读取出来。这个时候属于顺序读取，而不是磁盘寻道了，加快了速度。

支持范围查询，而且部分范围查询非常高效，每个节点能索引的范围更大更精确,也意味着 B+树单次磁盘IO的信息量大于B-树,I/O效率更高。

原因是数据都是存储在叶子节点这一层，并且有指针指向其他叶子节点，这样范围查询只需要遍历叶子节点这一层，无需整棵树遍历。

由于磁盘的存取速度与内存之间鸿沟,为了提高效率,要尽量减少磁盘I/O.磁盘往往不是严格按需读取，而是每次都会预读,磁盘读取完需要的数据,会顺序向后读一定长度的数据放入内存。而这样做的理论依据是计算机科学中著名的局部性原理：

当一个数据被用到时，其附近的数据也通常会马上被使用，程序运行期间所需要的数据通常比较集中

延伸问题：与哈希索引的区别？

简单地说，哈希索引就是采用一定的哈希算法，把键值换算成新的哈希值，检索时不需要类似B+树那样从根节点到叶子节点逐级查找，只需一次哈希算法即可立刻定位到相应的位置，速度非常快。

如果是等值查询，那么哈希索引明显有绝对优势，因为只需要经过一次算法即可找到相应的键值；当然了，这个前提是，键值都是唯一的。如果键值不是唯一的，就需要先找到该键所在位置，然后再根据链表往后扫描，直到找到相应的数据。

如果是范围查询检索，这时候哈希索引就毫无用武之地了，因为原先是有序的键值，经过哈希算法后，有可能变成不连续的了，就没办法再利用索引完成范围查询检索。

同理，哈希索引也没办法利用索引完成排序，以及like ‘xxx%’ 这样的部分模糊查询（这种部分模糊查询，其实本质上也是范围查询）。

哈希索引也不支持多列联合索引的最左匹配规则。

B+树索引的关键字检索效率比较平均，不像B树那样波动幅度大，在有大量重复键值情况下，哈希索引的效率也是极低的，因为存在所谓的哈希碰撞问题。

问题：MyISAM 和 InnoDB 的区别有哪些？

InnoDB 支持事务，但是MyISAM 不支持事务。这也是 MySQL 选择 InnoDB作为 默认存储引擎的原因之一；

InnoDB 支持外键，但是 MyISAM 不支持。如果一个表包含外键，并且存储引擎是InnoDB，把它转为 MyISAM就会失败；

InnoDB 使用的是聚集索引，MyISAM使用非聚集索引。聚簇索引的文件存放在主键索引的叶子节点上，所以 InnoDB 必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。所以，主键不应该过大，因为主键太大，其他索引也都会很大。非聚集索引的话，数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。

InnoDB 不保存表的具体行数，执行 select count(\*) from table 时需要全表扫描。但是MyISAM用一个变量保存了整个表的行数，执行上述语句时只需要读出该变量即可，速度很快；

InnoDB 最小的锁粒度是行级锁，MyISAM 最小的锁粒度是表级锁。一个更新语句会锁住整张表，导致其他查询和更新都会被阻塞，所以并发访问受到很大的限制。

问题：延伸问题：如何选择存储引擎？

是否要支持事务，如果要请选择 InnoDB，如果不需要可以考虑 MyISAM；

如果表中绝大多数都只是读查询，可以考虑 MyISAM，如果既有读写也挺频繁，那就使用InnoDB。

系统崩溃后，MyISAM恢复起来更困难，能否接受，不能接受就选 InnoDB；

MySQL5.5版本开始InnoDB已经成为MySQL的默认引擎，说明其优势是有目共睹的。如果不知道用什么存储引擎，那就用InnoDB，跟着官方走，至少不会差。

问题：MySQL主从复制是怎么做的？

主从复制主要涉及三个线程：binlog 线程、I/O 线程和 SQL 线程。这个过程是靠这三个过程的密切配合来进行的。

binlog 线程 ：负责将主服务器上的数据更改写入二进制日志（Binary log）中。

I/O 线程 ：负责从主服务器上读取二进制日志，并写入从服务器的中继日志（Relay log）。

SQL 线程 ：负责读取中继日志，解析出主服务器已经执行的数据更改并在从服务器中重放（Replay）。

问题：大表优化

当MySQL单表记录数过大时，数据库的CRUD性能会明显下降，一些常见的优化措施如下：

限定数据的范围

务必禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内；

读/写分离

经典的数据库拆分方案，主库负责写，从库负责读；

垂直分区

根据数据库里面数据表的相关性进行拆分。 例如，用户表中既有用户的登录信息又有用户的基本信息，可以将用户表拆分成两个单独的表，甚至放到单独的库做分库。

简单来说垂直拆分是指数据表列的拆分，把一张列比较多的表拆分为多张表。

垂直拆分的优点： 可以使得列数据变小，在查询时减少读取的Block数，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。

垂直拆分的缺点： 主键会出现冗余，需要管理冗余列，并会引起Join操作，可以通过在应用层进行Join来解决。此外，垂直分区会让事务变得更加复杂；

4. 水平分区

保持数据表结构不变，通过某种策略存储数据分片。这样每一片数据分散到不同的表或者库中，达到了分布式的目的。 水平拆分可以支撑非常大的数据量。

水平拆分是指数据表行的拆分，表的行数超过200万行时，就会变慢，这时可以把一张的表的数据拆成多张表来存放。举个例子：我们可以将用户信息表拆分成多个用户信息表，这样就可以避免单一表数据量过大对性能造成影响。

水平拆分可以支持非常大的数据量。需要注意的一点是：分表仅仅是解决了单一表数据过大的问题，但由于表的数据还是在同一台机器上，其实对于提升MySQL并发能力没有什么意义，所以 水平拆分最好分库 。

水平拆分能够 支持非常大的数据量存储，应用端改造也少，但 分片事务难以解决 ，跨节点Join性能较差，逻辑复杂

问题：如何保证缓存与数据库双写时的数据一致性?

一般来说，就是如果系统不是严格要求缓存和数据库必须一致性的话，缓存可以稍微的跟数据库偶尔有不一致的情况，最好不要做这个方案，可以将读请求和写请求串行化，串到一个内存队列里去，这样就可以保证一定不会出现不一致的情况

串行化之后，就会导致系统的吞吐量会大幅度的降低，用比正常情况下多几倍的机器去支撑线上的一个请求。

问题：红黑树和AVL树有什么区别？

AVL 和RBT 都是二叉查找树的优化。其性能要远远好于二叉查找树。他们之间都有自己的优势，其应用上也有不同。

结构对比： AVL的结构高度平衡，RBT的结构基本平衡。平衡度AVL > RBT.

查找对比： AVL 查找时间复杂度最好，最坏情况都是O(logN)。

RBT 查找时间复杂度最好为O(logN)，最坏情况下比AVL略差。

插入删除对比： 1. AVL的插入和删除结点很容易造成树结构的不平衡，而RBT的平衡度要求较低。因此在大量数据插入的情况下，RBT需要通过旋转变色操作来重新达到平衡的频度要小于AVL。

如果需要平衡处理时，RBT比AVL多一种变色操作，而且变色的时间复杂度在O(logN)数量级上。但是由于操作简单，所以在实践中这种变色仍然是非常快速的。

当插入一个结点都引起了树的不平衡，AVL和RBT都最多需要2次旋转操作。但删除一个结点引起不平衡后，AVL最多需要logN 次旋转操作，而RBT最多只需要3次。因此两者插入一个结点的代价差不多，但删除一个结点的代价RBT要低一些。

AVL和RBT的插入删除代价主要还是消耗在查找待操作的结点上。因此时间复杂度基本上都是与O(logN) 成正比的。

总体评价：大量数据实践证明，RBT的总体统计性能要好于平衡二叉树。

问题：Redis 和 Memcached 的区别

Redis支持更丰富的数据类型（支持更复杂的应用场景）：Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。memcached只支持简单的字符串类型。

Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用，而Memecached把数据全部存在内存之中。

集群模式：Memcached没有原生的集群模式，需要依靠客户端来实现往集群中分片写入数据；但是 Redis 目前是原生支持 cluster 模式的.

Memcached是多线程，非阻塞I/O复用的网络模型；Redis使用单线程的多路 I/O 复用模型。

Redis 常见数据结构以及使用场景分析

String

String数据结构是简单的key-value类型，value其实不仅可以是String，也可以是数字。

可以用作常规key-value缓存，也可以用来计数，比如说记录微博数，粉丝数等。

Hash

hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象，后续操作的时候，可以直接仅仅修改这个对象中的某个字段的值。 比如我们可以 hash 数据结构来存储用户信息，商品信息等等。

List

list 就是链表，list的应用场景非常多，也是Redis最重要的数据结构之一，比如微博的关注列表，粉丝列表，消息列表等功能都可以用Redis的 list 结构来实现。list的底层是一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销。

Set

set 对外提供的功能与list类似是一个列表的功能，特殊之处在于 set 是可以自动排重的。

当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。可以基于 set 轻易实现交集、并集、差集的操作。

比如：在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。Redis可以非常方便的实现如共同关注、共同粉丝、共同喜好等功能。这个过程也就是求交集的过程。

Sorted Set

和set相比，sorted set增加了一个权重参数score，使得集合中的元素能够按score进行有序排列。

举例： 在直播系统中，实时排行信息包含直播间在线用户列表，各种礼物排行榜，弹幕消息（可以理解为按消息维度的消息排行榜）等信息，适合使用 Redis 中的 Sorted Set 结构进行存储。

问题：什么是AOF重写？

AOF重写可以产生一个新的AOF文件，这个新的AOF文件和原有的AOF文件所保存的数据库状态一样，但体积更小。

在执行 BGREWRITEAOF 命令时，Redis 服务器会维护一个 AOF 重写缓冲区，该缓冲区会在子进程创建新AOF文件期间，记录服务器执行的所有写命令。当子进程完成创建新AOF文件的工作之后，服务器会将重写缓冲区中的所有内容追加到新AOF文件的末尾，使得新旧两个AOF文件所保存的数据库状态一致。最后，服务器用新的AOF文件替换旧的AOF文件，以此来完成AOF文件重写操作。

缓存雪崩和缓存穿透问题解决方案

缓存雪崩

就是缓存同一时间大面积的失效，所以，后面的请求都会落到数据库上，造成数据库短时间内承受大量请求而崩掉。

解决办法：

事前：尽量保证整个 Redis 集群的高可用性，发现机器宕机尽快补上。选择合适的内存淘汰策略。

事中：本地缓存 + Hystrix限流和降级，避免MySQL崩掉。

事后：利用 Redis 持久化机制保存的数据尽快恢复缓存。

缓存穿透

一般是黑客故意去请求缓存中不存在的数据，导致所有的请求都落到数据库上，造成数据库短时间内承受大量请求而崩掉。

解决办法：

有很多种方法可以有效地解决缓存穿透问题，最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被 这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。另外也有一个更为简单粗暴的方法（我们采用的就是这种），如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

问题：数据库与数据仓库的区别

简单理解下数据仓库是多个数据库以一种方式组织起来

数据库强调范式，尽可能减少冗余

数据仓库强调查询分析的速度，优化读取操作，主要目的是快速做大量数据的查询

数据仓库定期写入新数据，但不覆盖原有数据，而是给数据加上时间戳标签

数据库采用行存储，数据仓库一般采用列存储（行存储与列存储区别见题3）

数据仓库的特征是面向主题、集成、相对稳定、反映历史变化，存储数历史数据；数据库是面向事务的，存储在线交易数据

数据仓库的两个基本元素是维表和事实表，维是看待问题的角度，比如时间、部门等，事实表放着要查询的数据

问题：SQL的数据类型

字符串：char、varchar、text

二进制串：binary、varbinary

布尔类型：boolean

数值类型：integer、smallint、bigint、decimal、numeric、float、real、double

时间类型：date、time、timestamp、interval

问题：left join,right join,inner join,full join之间的区别？

1.inner join（内连接），在两张表进行连接查询时，只保留两张表中完全匹配的结果集。

2.left join,在两张表进行连接查询时，会返回左表所有的行，即使在右表中没有匹配的记录。

3.right join,在两张表进行连接查询时，会返回右表所有的行，即使在左表中没有匹配的记录。

4.full join,在两张表进行连接查询时，返回左表和右表中所有没有匹配的行。

问题：having和where的区别？

本质的区别就是：

where筛选的是数据库表里面本来就有的字段，

而having筛选的字段是从前筛选的字段筛选的。

where和having都可以使用的场景：

select goods\_price,goods\_name from sw\_goods where goods\_price>100

select goods\_price,goods\_name from sw\_goods having goods\_price>100

1

2

3

原因：goods\_price作为条件也出现在了查询字段中。

只可以使用where，不可以使用having的情况：

select goods\_name,goods\_number from sw\_goods where goods\_price>100

select goods\_name,goods\_number from sw\_goods having goods\_price>100(X)

1

2

3

原因：goods\_price作为筛选条件没有出现在查询字段中，所以就会报错。

having的原理是先select 然后从select出来的进行筛选。而where是先筛选在select。

只可以使用having，不可以使用where的情况：

select goods\_category\_id,avg(good\_price) as ag from sw\_goods group by goods\_category having ag>1000

select goods\_category\_id,avg(goods\_price) as ag from sw\_goods where ag>1000 group by goods\_category(X)报错，这个表里没有这个ag这个字段。

1

2

3

where子句中一般不使用聚合函数那种情况。

问题：not in和not exists区别

如果查询语句使用了not in 那么内外表都进行全表扫描，没有用到索引；

而not extsts 的子查询依然能用到表上的索引。

所以无论那个表大，用not exists都比not in要快。

也就是说，in和exists需要具体情况具体分析，not in和not exists就不用分析了，尽量用not exists就好了。

问题：mysql中设置row number

SET @row\_number = 0;

SELECT (@row\_number:=@row\_number + 1) AS num FROM table

1

2

问题：sql中null与‘ ’的区别。

null表示空，用is null判断

'‘表示空字符串，用=’'判断

问题：mysql 中视图和表的区别以及联系是什么？

一、两者的区别

1）本质

表是内容，视图是窗口。视图是已经编译好的sql语句，是基于sql语句的结果集的可视化的表，而表不是。

2）实与虚

表属于全局模式中的表，是实表；视图属于局部模式的表，是虚表。

3）是否存在物理记录

视图没有，而表有。

4）是否占用物理空间

表占用物理空间，而视图不占用。视图只是逻辑概念的存在，表可以及时对它进行修改，但视图只能用创建的语句来修改。

5）是否影响

视图的建立（create）和删除（drop）只影响视图本身，不影响对应的基本表。

6）安全因素

视图是查看数据表的一种方法，可以查询数据表中某些字段构成的数据，只是一些sql语句的集合。从安全的角度来说，视图可以防止用户接触数据表，因而用户不知道表结构。

二、两者的联系

视图是在基本表之上建立的表，它的结构（即所定义的列）和内容（即所有记录）都来自基本表，它依据基本表存在而存在。一个视图可以对应一个基本表，也可以对应多个基本表。视图是基本表的抽象和在逻辑意义上建立的新关系。

问题：行存储和列存储的区别。

（1）行存储：传统数据库的存储方式，同一张表内的数据放在一起，插入更新很快。缺点是每次查询即使只涉及几列，也要把所有数据读取.

（2）列存储：OLAP等情况下，将数据按照列存储会更高效，每一列都可以成为索引，投影很高效。缺点是查询是选择完成时，需要对选择的列进行重新组装。

“当你的核心业务是 OLTP 时，一个行式数据库，再加上优化操作，可能是个最好的选择。

当你的核心业务是 OLAP 时，一个列式数据库，绝对是更好的选择”

问题：如何写SQL求出中位数平均数和众数（除了用count之外的方法）

1）中位数

方案1（没考虑到偶数个数的情况）：

set @m = (select count(\*)/2 from table)

select column from table order by column limit @m, 1

方案2（考虑偶数个数，中位数是中间两个数的平均）：

set @index = -1

select avg(table.column)

from

(select @index:=@index+1 as index, column

from table order by column) as t

where t.index in (floor(@index/2),ceiling(@index/2))

2）平均数：select avg(distinct column) from table

3）众数：select column, count(\*) from table group by column order by column desc limit 1

4、机器学习问题

问题：如何避免决策树过拟合？

1）限制树深

2）剪枝

3）限制叶节点数量

4）正则化项

5）增加数据

6）bagging（subsample、subfeature、低维空间投影）

7）数据增强（加入有杂质的数据）

8）早停

问题：请说明随机森林较一般决策树稳定的几点原因

1）bagging的方法，多个树投票提高泛化能力

2）bagging中引入随机（参数、样本、特征、空间映射），避免单棵树的过拟合，提高整体泛化能力

问题：SVM的优点

1）优点：

a. 能应用于非线性可分的情况

b. 最后分类时由支持向量决定，复杂度取决于支持向量的数目而不是样本空间的维度，避免了维度灾难

c. 具有鲁棒性：因为只使用少量支持向量，抓住关键样本，剔除冗余样本

d. 高维低样本下性能好，如文本分类

1

2

3

4

5

6

7

2）缺点：

a. 模型训练复杂度高

b. 难以适应多分类问题

c. 核函数选择没有较好的方法论

1

2

3

4

5

问题：K-means的原理

1）初始化k个点

2）根据距离点归入k个类中

3）更新k个类的类中心

4）重复（2）（3），直到收敛或达到迭代次数

问题：K-Means算法原理及改进，遇到异常值怎么办？评估算法的指标有哪些？

k-means原理：

在给定K值和K个初始类簇中心点的情况下，把每个点(亦即数据记录)分到离其最近的类簇中心点所代表的类簇中，所有点分配完毕之后，根据一个类簇内的所有点重新计算该类簇的中心点(取平均值)，然后再迭代的进行分配点和更新类簇中心点的步骤，直至类簇中心点的变化很小，或者达到指定的迭代次数。

改进：

a. kmeans++：初始随机点选择尽可能远，避免陷入局部解。方法是n+1个中心点选择时，对于离前n个点选择到的概率更大

b. mini batch kmeans：每次只用一个子集做重入类并找到类心（提高训练速度）

c. ISODATA：对于难以确定k的时候，使用该方法。思路是当类下的样本小时，剔除；类下样本数量多时，拆分

d. kernel kmeans：kmeans用欧氏距离计算相似度，也可以使用kernel映射到高维空间再聚类

遇到异常值：

a. 有条件的话使用密度聚类或者一些软聚类的方式先聚类，剔除异常值。不过本来用kmeans就是为了快，这么做有些南辕北辙了

b. 局部异常因子LOF：如果点p的密度明显小于其邻域点的密度，那么点p可能是异常值

（参考：https://blog.csdn.net/wangyibo0201/article/details/51705966）

c. 多元高斯分布异常点检测

d. 使用PCA或自动编码机进行异常点检测：使用降维后的维度作为新的特征空间，其降维结果可以认为剔除了异常值的影响（因为过程是保留使投影后方差最大的投影方向）

e. isolation forest：基本思路是建立树模型，一个节点所在的树深度越低，说明将其从样本空间划分出去越容易，因此越可能是异常值。是一种无监督的方法，随机选择n个sumsampe，随机选择一个特征一个值。

（参考：https://blog.csdn.net/u013709270/article/details/73436588）

f. winsorize：对于简单的，可以对单一维度做上下截取

评估聚类算法的指标：

a. 外部法（基于有标注）：Jaccard系数、纯度

b. 内部法（无标注）：内平方和WSS和外平方和BSS

c. 此外还要考虑到算法的时间空间复杂度、聚类稳定性等

问题：PCA（主成分分析）

主成分分析是一种降维的方法

思想是将样本从原来的特征空间转化到新的特征空间，并且样本在新特征空间坐标轴上的投影方差尽可能大，这样就能涵盖样本最主要的信息

方法：

a. 特征归一化

b. 求样本特征的协方差矩阵A

c. 求A的特征值和特征向量，即AX=λX

d. 将特征值从大到小排列，选择topK，对应的特征向量就是新的坐标轴（采用最大方差理论解释，参考：

问题：时间序列分析

问题：数据预处理过程有哪些？

缺失值处理：删、插

异常值处理

特征转换：时间特征sin化表示

标准化：最大最小标准化、z标准化等

归一化：对于文本或评分特征，不同样本之间可能有整体上的差异，如a文本共20个词，b文本30000个词，b文本中各个维度上的频次都很可能远远高于a文本

离散化：onehot、分箱等

问题：数据清理中，处理缺失值的方法是？

由于调查、编码和录入误差，数据中可能存在一些无效值和缺失值，需要给予适当的处理。常用的处理方法有：估算，整例删除，变量删除和成对删除。

（1）估算(estimation)。最简单的办法就是用某个变量的样本均值、中位数或众数代替无效值和缺失值。这种办法简单，但没有充分考虑数据中已有的信息，误差可能较大。另一种办法就是根据调查对象对其他问题的答案，通过变量之间的相关分析或逻辑推论进行估计。例如，某一产品的拥有情况可能与家庭收入有关，可以根据调查对象的家庭收入推算拥有这一产品的可能性。

（2）整例删除(casewise deletion)是剔除含有缺失值的样本。由于很多问卷都可能存在缺失值，这种做法的结果可能导致有效样本量大大减少，无法充分利用已经收集到的数据。因此，只适合关键变量缺失，或者含有无效值或缺失值的样本比重很小的情况。

（3）)变量删除(variable deletion)。如果某一变量的无效值和缺失值很多，而且该变量对于所研究的问题不是特别重要，则可以考虑将该变量删除。这种做法减少了供分析用的变量数目，但没有改变样本量。

（4）成对删除(pairwise deletion)是用一个特殊码(通常是9、99、999等)代表无效值和缺失值，同时保留数据集中的全部变量和样本。但是，在具体计算时只采用有完整答案的样本，因而不同的分析因涉及的变量不同，其有效样本量也会有所不同。这是一种保守的处理方法，最大限度地保留了数据集中的可用信息。

问题： 余弦距离与欧式距离求相似度的差别？

欧氏距离能够体现个体数值特征的绝对差异，所以更多的用于需要从维度的数值大小中体现差异的分析，如使用用户行为指标分析用户价值的相似度或差异。

余弦距离更多的是从方向上区分差异，而对绝对的数值不敏感，更多的用于使用用户对内容评分来区分兴趣的相似度和差异，同时修正了用户间可能存在的度量标准不统一的问题（因为余弦距离对绝对数值不敏感）。

总体来说，欧氏距离体现数值上的绝对差异，而余弦距离体现方向上的相对差异

（1）例如，统计两部剧的用户观看行为，用户A的观看向量为(0,1)，用户B为(1,0)；此时二者的余弦距很大，而欧氏距离很小；我们分析两个用户对于不同视频的偏好，更关注相对差异，显然应当使用余弦距离。

（2）而当我们分析用户活跃度，以登陆次数(单位：次)和平均观看时长(单：分钟)作为特征时，余弦距离会认为(1,10)、(10,100)两个用户距离很近；但显然这两个用户活跃度是有着极大差异的，此时我们更关注数值绝对差异，应当使用欧氏距离。

问题：分类算法性能的主要评价指标

查准率、查全率、F1

AUC

LOSS

Gain和Lift

WOE和IV

问题： GBDT（梯度提升树）

首先介绍Adaboost Tree，是一种boosting的树集成方法。基本思路是依次训练多棵树，每棵树训练时对分错的样本进行加权。树模型中对样本的加权实际是对样本采样几率的加权，在进行有放回抽样时，分错的样本更有可能被抽到

GBDT是Adaboost Tree的改进，每棵树都是CART（分类回归树），树在叶节点输出的是一个数值，分类误差就是真实值减去叶节点的输出值，得到残差。GBDT要做的就是使用梯度下降的方法减少分类误差值。

在GBDT的迭代中，假设我们前一轮迭代得到的强学习器是ft−1(x), 损失函数是L(y,ft−1(x)), 我们本轮迭代的目标是找到一个CART回归树模型的弱学习器ht(x)，让本轮的损失损失L(y,ft(x)=L(y,ft−1(x)+ht(x))最小。也就是说，本轮迭代找到决策树，要让样本的损失尽量变得更小。

GBDT的思想可以用一个通俗的例子解释，假如有个人30岁，我们首先用20岁去拟合，发现损失有10岁，这时我们用6岁去拟合剩下的损失，发现差距还有4岁，第三轮我们用3岁拟合剩下的差距，差距就只有一岁了。如果我们的迭代轮数还没有完，可以继续迭代下面，每一轮迭代，拟合的岁数误差都会减小。

（参考：https://www.cnblogs.com/pinard/p/6140514.html）

得到多棵树后，根据每颗树的分类误差进行加权投票

5、业务场景问题

问题：处理需求时的一般思路是什么？，并举例

1）明确需求，需求方的目的是什么

2）拆解任务

3）制定可执行方案

4）推进

5）验收

问题：业务场景题，如何分析次日留存率下降的问题？

业务问题关键是问对问题，然后才是拆解问题去解决。

1）两层模型：从用户画像、渠道、产品、行为环节等角度细分，明确到底是哪里的次日留存率下降了

2）指标拆解：次日留存率 = Σ 次日留存数 / 今日获客人数

3）原因分析：

（1）内部：

a. 运营活动

b. 产品变动

c. 技术故障

d. 设计漏洞（如产生可以撸羊毛的设计）

（2）外部：

a. 竞品

b. 用户偏好

c. 节假日

d. 社会事件（如产生舆论）

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

问题：给你一个无序数组，怎么才能合理采样？

无序数组是相对有序数组而言的，无序数组并不等于随机，我们要做的是将无序数组洗牌，得到随机排列。

对于无序数组，n个元素能产生n！种排序。如果洗牌算法能产生n！种不同的结果，并且这些结果产生的概率相等，那么这个洗牌算法是正确的。

方法：for i in range(len(n)): swap(arr[i], arr[random(i,n)])

这段代码是对随机确定数组第一位的值，然后递归对剩余的数组进行相同的过程，可以产生n！中等可能的排序情况。

问题：如果次日用户留存率下降了 5%该怎么分析？

1）首先采用“两层模型”分析：对用户进行细分，包括新老、渠道、活动、画像等多个维度，然后分别计算每个维度下不同用户的次日留存率。通过这种方法定位到导致留存率下降的用户群体是谁

2）对于目标群体次日留存下降问题，具体情况具体分析。具体分析可以采用“内部-外部”因素考虑，内部因素分为获客（渠道质量低、活动获取非目标用户）、满足需求（新功能改动引发某类用户不满）、提活手段（签到等提活手段没打成目标、产品自然使用周期低导致上次获得的大量用户短期内不需要再使用等）；外部因素采用PEST分析，政治（政策影响）、经济（短期内主要是竞争环境，如对竞争对手的活动）、社会（舆论压力、用户生活方式变化、消费心理变化、价值观变化等偏好变化）、技术（创新解决方案的出现、分销渠道变化等）

问题： 卖玉米如何提高收益？价格提高多少才能获取最大收益？

收益 = 单价\*销售量，那么我们的策略是提高单位溢价或者提高销售规模。

提高单位溢价的方法：

（1）品牌打造获得长期溢价，但缺陷是需要大量前期营销投入；

（2）加工商品占据价值链更多环节，如熟玉米、玉米汁、玉米蛋白粉；重定位商品，如礼品化等；

（3）价格歧视，根据价格敏感度对不同用户采用不同定价。

销售量=流量x转化率，上述提高单位溢价的方法可能对流量产生影响，也可能对转化率产生影响。

收益 = 单价x流量x转化率，短期内能规模化采用的应该是进行价格歧视，如不同时间、不同商圈的玉米价格不同，采取高定价，然后对价格敏感的用户提供优惠券等。

问题：怎么做恶意刷单检测?

分类问题用机器学习方法建模解决，我想到的特征有：

（1）商家特征：商家历史销量、信用、产品类别、发货快递公司等

（2）用户行为特征：用户信用、下单量、转化率、下单路径、浏览店铺行为、支付账号

（3）环境特征（主要是避免机器刷单）：地区、ip、手机型号等

（4）异常检测：ip地址经常变动、经常清空cookie信息、账号近期交易成功率上升等

（5）评论文本检测：刷单的评论文本可能套路较为一致，计算与已标注评论文本的相似度作为特征

（6）图片相似度检测：同理，刷单可能重复利用图片进行评论

问题：类比到头条的收益，头条放多少广告可以获得最大收益，不需要真的计算，只要有个思路就行？

收益 = 出价x流量x点击率x有效转化率，放广告的数量会提高流量，但会降低匹配程度，因此降低点击率。最大收益是找到这个乘积的最大值，是一个有约束条件的最优化问题。

同时参考价格歧视方案，可以对不同的用户投放不同数量的广告。

费米估计

参考

问题：不用任何公开参考资料，估算今年新生儿出生数量

1）采用两层模型（人群画像人群转化）：新生儿出生数=Σ各年龄层育龄女性数量各年龄层生育比率

2）从数字到数字：如果有前几年新生儿出生数量数据，建立时间序列模型（需要考虑到二胎放开的突变事件）进行预测

3）找先兆指标，如婴儿类用品的新增活跃用户数量X表示新生儿家庭用户。Xn/新生儿n为该年新生儿家庭用户的转化率，如X2007/新生儿2007位为2007年新生儿家庭用户的转化率。该转化率会随平台发展而发展，可以根据往年数量推出今年的大致转化率，并根据今年新增新生儿家庭用户数量推出今年估计的新生儿数量。

AB测试

问题：A/B test是什么？

A / B测试（也称为分割测试或桶测试）是一种将网页或应用程序的两个版本相互比较以确定哪个版本的性能更好的方法。AB测试本质上是一个实验，其中页面的两个或多个变体随机显示给用户，统计分析确定哪个变体对于给定的转换目标（指标如CTR）效果更好。

问题：A/B test工作原理

在A / B test中，你可以设置访问网页或应用程序屏幕并对其进行修改以创建同一页面的第二个版本。这个更改可以像单个标题或按钮一样简单，也可以是完整的页面重新设计。然后，一半的流量显示页面的原始版本（称为控件），另一半显示页面的修改版本（称为变体）。

当用户访问页面时，如上图灰色按钮（控件）和箭头所指红色按钮（变体），利用埋点可以对用户点击行为数据采集，并通过统计引擎进行分析（进行A/B test）。然后，就可以确定这种更改（变体）对于给定的指标（这里是用户点击率CTR）产生正向影响，负向影响或无影响。

实验数据结果可能如下：

问题：进行A/B test的目的是什么？

A / B test可以让个人，团队和公司通过用户行为结果数据不断对其用户体验进行仔细更改。这允许他们构建假设，并更好地了解为什么修改的某些元素会影响用户行为。这些假设可能被证明是错误的，也就是说他们对特定目标的最佳体验的个人或团队想法利用A / B test证明对用户来说是行不通的，当然也可能证明是正确的。

所以说 A/B test不仅仅是解决一次分歧的对比，A/B test可以持续使用，以不断改善用户的体验，改善某一目标，如随着时间推移的转换率。

问题：A/B test流程

①确定目标：目标是用于确定变体是否比原始版本更成功的指标。可以是点击按钮的点击率、链接到产品购买的打开率、电子邮件注册的注册率等等。

②创建变体：对网站原有版本的元素进行所需的更改。可能是更改按钮的颜色，交换页面上元素的顺序，隐藏导航元素或完全自定义的内容。

③生成假设：一旦确定了目标，就可以开始生成A / B测试想法和假设，以便统计分析它们是否会优于当前版本。

④收集数据：针对指定区域的假设收集相对应的数据用于A/B test分析。

⑤运行试验：此时，网站或应用的访问者将被随机分配控件或变体。测量，计算和比较他们与每种体验的相互作用，以确定每个用户体验的表现。

⑥分析结果：实验完成后，就可以分析结果了。A / B test分析将显示两个版本之间是否存在统计性显著差异。

问题：A/B test需要注意的点

1、先验性：通过低代价，小流量的实验，在推广到全流量的用户。

2、并行性：不同版本、不同方案在验证时，要保重其他条件都一致。

3、分流科学性和数据科学性：分流科学是指对AB两组分配的数据要一致，数据科学性是指不能直接用均值转化率、均值点击率来进行AB test决策，而是要通过置信区间、假设检验、收敛程度来得出结论。

问题：A/B test中要知道的统计学知识

1、点估计

2、区间估计

3、中心极限定理（样本估计总体的核心，可以对比看一下大数定理）

4、假设检验

其中假设检验部分为核心，其他辅助更好的理解该部分内容，比如区间估计可以理解为正向的推断统计，假设检验可以理解为反证的推断统计，关于假设检验本身，你可能还需要知道小概率事件、t分布、z分布、卡方分布、p值、alpha错误、belta错误等内容。

问题：A/B test简例（结合Python实现）

实例背景简述：

某司「猜你想看」业务接入了的新推荐算法，新推荐策略算法开发完成后，在全流量上线之前要评估新推荐策略的优劣，所用的评估方法是A/B test，具体做法是在全量中抽样出两份小流量，分别走新推荐策略分支和旧推荐策略分支，通过对比这两份流量下的指标（这里按用户点击衡量）的差异，可以评估出新策略的优劣，进而决定新策略是否全适合全流量。

实例A/B test步骤：

指标：CTR

变体：新的推荐策略

假设：新的推荐策略可以带来更多的用户点击。

收集数据：以下B组数据为我们想验证的新的策略结果数据，A组数据为旧的策略结果数据。均为伪造数据。

分析结果（Python）：

\*\*利用 python 中的 scipy.stats.ttest\_ind 做关于两组数据的双边 t 检验，结果比较简单。\*\*但是做大于或者小于的单边检测的时候需要做一些处理，才能得到正确的结果。

from scipy import stats

import numpy as np

import numpy as np

import seaborn as sns

A = np.array([ 1, 4, 2, 3, 5, 5, 5, 7, 8, 9,10,18])

B = np.array([ 1, 2, 5, 6, 8, 10, 13, 14, 17, 20,13,8])

print('策略A的均值是：',np.mean(A))

print('策略B的均值是：',np.mean(B))

1

2

3

4

5

6

7

8

9

10

Output:

策略A的均值是：6.416666666666667

策略B的均值是：9.75

1

2

3

很明显，策略B的均值大于策略A的均值，但这就能说明策略B可以带来更多的业务转化吗？还是说仅仅是由于一些随机的因素造成的。

我们是想证明新开发的策略B效果更好，所以可以设置原假设和备择假设分别是:

H0：A>=B

H1：A < B

scipy.stats.ttest\_ind(x,y)默认验证的是x.mean()-y.mean()这个假设。为了在结果中得到正数，计算如下：

stats.ttest\_ind(B,A,equal\_var= False)

1

output:

Ttest\_indResult(statistic=1.556783470104261, pvalue=0.13462981561745652)

1

2

根据 scipy.stats.ttest\_ind(x, y) 文档的解释，这是双边检验的结果。为了得到单边检验的结果，需要将 计算出来的 pvalue 除于2 取单边的结果(这里取阈值为0.05）。

求得pvalue=0.13462981561745652，p/2 > alpha(0.05),所以不能够拒绝假设，暂时不能够认为策略B能带来多的用户点击。

数据埋点

问题：什么是埋点？

数据埋点是一种移动端APP常规的数据采集方法

埋点是数据采集的一种方法，将移动APP 每个功能需要统计的点击行为、页面上的功能使用情况，采集相应的信息和行为。无论是产品的迭代还是运营的策略，都是需要有详细的数据支撑来针对性的做下一步迭代和运营的决策。有了数据分析，你可以得到用户画像、用户行为路径，不用再去做大量用户调研、盲目的猜原因，为我们大大降低了试错的成本。

问题：埋点方式有哪些？

埋点方式从数据的来源分为客户端埋点和服务端埋点

客户端埋点理解为用户行为操作的数据采集，服务端是用户通过客户端发生请求获取反馈的数据采集，选择不同方式的场景主要涉及哪些呢，譬如我们在手机APP端频繁的操作刷新、点击、返回，这些操作行为的数据大多数采用客户端埋点方式，适用于大量频繁的操作并不需要实时反馈信息的场景，同时客户端具有缓存的功能，这样的埋点方式不仅对客户的产品体验好，可以减轻服务器端的信息交互压力

服务端埋点更使用与交互少，数据反馈要求实时性高，比如新闻信息的变化，比如答题的答案选项、对错情况。

客户端埋点：

优：采集的APP端页面展示、点击行为，不需要请求服务器的数据

缺：无网络时数据不完整、实时性有延迟；当需要改变埋点时，必须更新版本。

服务端埋点：

优：

（1）实时性好，数据准确；

（2）变更成本低；

（3）能够收集不在APP内发生的行为，只要请求服务器就行。如统计从其他APP引流的安装量。

缺：不能收集不需要请求服务器的数据；用户不联网不能采集数据

如何评估一个活动

问题：评估一个活动的思路

参考

6、Python问题

问题：常用的Python库有哪些？

1）numpy：矩阵运算

2）sklearn：常用机器学习和数据挖掘工具库

3）scipy：基于numpy做高效的数学计算，如积分、线性代数、稀疏矩阵等

4）pandas：将数据用表的形式进行操作

5）matplotlib：数据可视化工具

6）seaborn：数据可视化工具

7）keras/tensorflow/theano：深度学习工具包

8）NLTK：自然语言处理工具包

9）beautifulsoap：网页文档解析工具

7、大数据问题：

问题：hadoop原理和mapreduce原理

1）Hadoop原理：采用HDFS分布式存储文件，MapReduce分解计算

2）MapReduce原理：

a. map阶段：读取HDFS中的文件，解析成<k,v>的形式，并对<k,v>进行分区（默认一个区），将相同k的value放在一个集合中

b. reduce阶段：将map的输出copy到不同的reduce节点上，节点对map的输出进行合并、排序

select \*

from

(select a.id,b.name,b.weight,a.total

from (select id，goods\_id,sum(count) as total

from trans

group by goods\_id) a inner join goods b

on a.goods\_id = b.id) c

where c.total > 20 and c.weight < 50;

1

2

3

4

5

6

7

8

9

10

手写SQL

问题：有一张用户签到表【t\_user\_attendence】，标记每天用户是否签到（说明：该表包含所有用户所有工作日的出勤记录） ，包含三个字段：日期【fdate】，用户id【fuser\_id】，用户当天是否签到【fis\_sign\_in：0否1是】；

问题1：请计算截至当前每个用户已经连续签到的天数（输出表仅包含当天签到的所有用户，计算其连续签到天数）

输出表【t\_user\_consecutive\_days】:用户id【fuser\_id】，用户联系签到天数【fconsecutive\_days】

思路：先找用户最近一次未签到日期，再用今天减那个日期

create table t\_user\_consecutive\_days as

select fuser\_id

,datediff('20200322',fdate\_max) fconsecutive\_days

from

(select fuser\_id

,max(fdate) fdate\_max

from t\_user\_attendence

where fis\_sign\_in = 0

group by fuser\_id

) t1

;

1

2

3

4

5

6

7

8

9

10

11

问题2：请计算每个用户历史以来最大的连续签到天数（输出表为用户签到表中所有出现过的用户，计算其历史最大连续签到天数）

输出表【t\_user\_max\_days】:用户id【fuser\_id】，用户最大连续签到天数【fmax\_days】

问题2答案：把用户所有签到记录转化成一条0-1字符串序列，用0做split切割，计算切出来的1序列组中的最大长度

create table t\_user\_max\_days as

select fuser\_id

,max(length(cut\_fsign\_record)) as fmax\_days

(select fuser\_id

,fsign\_record

,cut\_fsign\_record

from

(select fuser\_id

,wm\_concat(fis\_sign\_in) fsign\_record

from t\_user\_attendence

group by fuser\_id

) t1

lateral view explode(split(fsign\_record,'0')) t as cut\_fsign\_record

) t2

where cut\_fsign\_record<>''

group by fuser\_id

;

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

手写Python

题目：针对股票的最大回撤率指标定义，给出代码实现思路。给定的是产品所有交易日的净值序列，且其净值序列已按照日期排序。

最大回撤率：在选定周期内任一历史时点往后推，产品净值走到最低点时的收益率回撤幅度的最大值。

追问：如何在提升计算效率？

这道题类似的题目其实在leecode也有，这个大概是变化但类似版本（可以搜leecode股票最大回报）；因为团队里处理比较多金融资产数据，这个指标是策略中最常见的指标之一，所以也是一道工作中攒下来的题目。这个指标的计算优化问题真的非常值得问，我后面会列几个版本的代码思路和实现代码。

通常最简单的计算实现，会需要O(n2)的计算复杂度；可以针对如何降低计算复杂度，专门追问。

基础实现

def max\_drawdown(accnavArr):

mdd = 0

for i in range(0, len(accnavArr)):

for j in range(i + 1, len(accnavArr)):

drawdown = accnavArr[i] / accnavArr[j] - 1

if drawdown < mdd:

mdd = drawdown

return mdd

1

2

3

4

5

6

7

8

空间换时间实现版本：

把每个时间点计算的最大值都存到一个列表结构中，最大回撤的计算只需要再依赖这个列表进行多一次循环计算。

def maxDrawdownGainCal(accnavArr): # 默认accnavArr按日期降序排列

maxDrawdown = 10000

maxGain =0

arr\_len = len(accnavArr)

maxList = [0.0] \* arr\_len

minList = [0.0] \* arr\_len

maxList[arr\_len-1] = accnavArr[arr\_len-1]

minList[arr\_len-1] = accnavArr[arr\_len-1]

for i in range(arr\_len-2,-1,-1):

if accnavArr[i] > maxList[i+1]:

maxList[i] = accnavArr[i]

else:

maxList[i] = maxList[i+1]

if accnavArr[i] < minList[i+1]:

minList[i] = accnavArr[i]

else:

minList[i] = minList[i+1]

for i in range(0,arr\_len):

mdd = (accnavArr[i]/maxList[i]-1)

mg = (accnavArr[i]/minList[i]-1)

if mdd < maxDrawdown : maxDrawdown = mdd

if mg > maxGain : maxGain = mg

return maxDrawdown,maxGain

————————————————