mysql总结

1.基础知识

ubuntu启动mysql服务

sudo service mysql start

查看是否启动成功（查看是否监听3306端口）

netstat -tulnp|grep 3306

或者使用 mysqladmin -uroot -p密码 ping

2.常用操作

select 查询表达式exp1[exp2,…]

[

from 表table

[where 条件]

[group by {列名|位置position} [ASC|DESC]]

[having 条件]

[order by {列名|表达式|位置} [ASC|DESC]]

[limit {[偏移量，]行数|行数偏移量 }]

]

3.常见面试题

1）主键、超键、外键和候选键

主键：数据库标准对存储数据对象予以唯一和完整标识的数据列或属性的组合，一个数据列只能有一个主键，且主键取值不能缺失，即不能为空值null。(用于区分两条不同的数据)

超键：在关系中能唯一标识元组的属性集的关系模式的超键。一个属性可以作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。

候选键：是最小超键，即没有冗余元素的超键。

外键：在一个表中存在的另一个表的主键称为此表的外键。

2）数据库事务的四个特性及含义

数据库事务（transaction），是指作为单个逻辑单元执行的一系列操作，要么完全执行，要么完全不执行。

数据库事务正确执行的四个基本要素。ACID，原子性（Atomicity）、一致性（Correspondence）、隔离性（Isolation）、持久性（Durability）。

原子性：整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。

隔离性：隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间混淆，必须串行化或序列化请求，使得在同一时间仅有一个请求用于同一数据。

持久性：在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

==============

脏读：在一个事务处理过程中读取了另一个未提交的事务中的数据。

幻读、不可重复度

幻读和不可重复读都是读取了另一条已经提交的事务，不可重复读重点在于update和delete，而幻读在于insert。

在可重复读中，sql第一次读取到数据后，就将这些数据加锁，其他事务无法修改这些数据，就可以实现重复读了。但是这种方法却无法锁住insert的数据，所以当事务a先前读取了数据，或者修改了全部数据，事务b还是可以insert数据提交，这时事务a就会发现多了一条之前没有的数据，这就是幻读，不可以通过加行锁来避免。需要serializable隔离级别，读用读锁，写用写锁，读锁和写锁互斥，这样就可以有效避免幻读、不可重复读和脏读等问题，但是会极大降低数据库的并发能力。

====================

mysql数据库提供的四种隔离级别：

1）serializable(串行化)：可避免脏读、不可重复读、幻读；

2）repeatable read（可重复读）：可避免脏读、不可重复读；（默认使用）

3）read committed（读已提交）：可避免脏读的发生；

4）read uncommitted（读未提交）：最低级别，任何情况都无法保证。

=======================

锁模式包括：

共享锁：（读取）操作创建的锁。其他用户可以并发读取数据，但任何事物都不能获取数据上的排他锁，直到已释放所有共享锁。

排他锁（X锁）：对数据A加上排他锁后，则其他事务不能再对A加任何类型的封锁。获准排他锁的事务既能读数据又能修改数据。

更新锁：更新锁可以防止通常形式的死锁。如果两个事务获得了资源上的共享锁，然后试图同时更新数据，则两个事务都需要转换共享锁为排他锁，并且每个事务都等待另一个事务释放共享锁，因此发生死锁。

若要避免这种潜在的死锁问题，请使用更新锁。一次只有一个事务可以获得资源的更新，如果事务修改资源，则更新锁转换为排他锁。否则锁转换为共享锁。

锁的粒度：

行锁：粒度最小，并发性高

页锁：一次锁定一页，25个行锁可以升级为页锁

表锁：粒度大，并发性低

数据库锁：控制整个数据库操作

乐观锁：相对悲观锁而言，乐观锁认为数据一般情况下不会造成冲突，所以在数据进行提交更新时，才会正式对数据的冲突与否进行检测，如果发现冲突，则返回用户错误的信息，让用户决定如何处理。一般的实现乐观锁的方式是记录数据版本。

悲观锁：每次去拿数据的时候都认为别人会修改，所以在拿数据时都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁。传统的关系型数据库里就用了很多这种锁机制。

3）视图的作用，视图可以更改吗？

视图是虚拟的表，与包含数据的表不一样，视图只包含使用时动态检索数据的查询；不包含任何列或数据。使用视图可以简化复杂的sql操作，隐藏具体的细节，保护数据；视图创建后，可以使用与表相同的方式利用它们。

视图是不能被索引，也不能有关联的触发器或默认值，如果视图本身内有order by则对视图再次order by将被覆盖。

创建视图：create view 视图名 as xxxxxxx;

对于某些视图比如未使用联接子查询分组聚集函数distinct Union等，是可以对其更新的，对视图的更新将对原始表进行更新；但是视图主要用于简化检索，保护数据，并不用于更新，而且大部分视图都不可以更新。

4）drop，delete与truncate的区别

drop直接删除表，truncate删除表中的数据（先删除后重建表，表中无数据），再插入时自增长id又从1开始，delete删除表中数据，可以加where子句。

(1)delete语句执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中以便进行回滚操作。truncate table则一次性地删除表中所有的数据并不把单独的删除操作记录到日志中，删除行是不可恢复的。并且在删除的过程中不会激活与表有关的删除触发器。执行速度快。

（2）表和索引所占空间。当表被truncate后，这个表和索引所占空间会恢复到初始大小，而delete操作不会减少表或索引所占用的空间。drop语句会将表所占用的空间全部释放掉。

（3）一般而言，drop>truncate>delete

（4）应用范围。truncate只针对表，delete可以是表和视图。

（5）truncate和delete只删除数据，而drop删除整个表（结构和数据）

（6）truncate与不带where的delete：只删除数据，而不删除表结构，drop语句将删除表结构被依赖的约束(constrain)，触发器(trigger)，索引(index)；依赖于该表的存储过程/函数将被保留，但其状态会变为invalid。

（7）delete语句为DML(data maintain language)，这个操作会被放到rollback segment中，事务提交后才生效。如果有相应的trigger，执行时被触发。

（8）truncate、drop是DLL(data define language)，操作立即生效，原数据不放到rollback segment中，不能回滚

（9）在没有备份情况下，谨慎使用drop与truncate。要删除部分数据行采样delete且注意结合where来约束影响范围。回滚段要足够大。要删除表用drop；若想保留表而将表中数据删除，如果与事务无关，用truncate即可实现，如果与事务有关，或想触发trigger，还是用delete。

（10）truncate table 表名 速度快，而且效率高，因为truncate table在功能上与不带where子句的delete相同，二者均删除表中的全部行，但truncate table比delete快，且使用的系统和事务日志资源少，delete语句每次删除一行，并在事务日志中为所删除的每行记录一项。truncate table通过释放存储表数据所用的数据页来删除数据，并且每次只在事务日志中记录页的释放。

（11）truncate table删除表中的所有行，但表的结构及其列、约束、索引等保持不变。新行标识所用的计数值重置为该列的种子。如果想保留标识计数值，请改用delete。如果要删除表定义及其数据，请使用drop table。

（12）对于由foreign key(外键)约束引用的表，不能使用truncate table，而使用不带where子句的delete语句。由于truncate table不记录在日志中，所以它不能激活触发器。

5）索引的工作原理及其种类

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树及其变种B+树。

在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法，这种数据结构就是索引。

为表设置索引要付出代价：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）。

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。

增加索引不利的方面。

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、删除和修改时，索引也要动态地维护，这样就降低了数据的维护速度。

索引是建立在数据库表中的某些列上。在创建索引时，应该考虑在哪些列上可以创建索引，在哪些列上不可以创建索引。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子句中的列上创建索引，加快条件的判断速度。

一般来说，不应该创建索引的列具有下列特点：

第一，对于那些在查询中很少使用或者参考的列不应该使用索引。

第二，对于那些只有很少数据值的列也不应该增加索引。

第三，对于那些定义为text，image和bit数据类型的列也不应该增加索引。

第四，当修改性能远大于检索性能时，不应该创建索引。

根据数据库的功能，可以在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。

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）顺序是相同的。一个表只能包含一个聚集索引。（逻辑顺序决定物理顺序）

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。

============================

局部性原理与磁盘预读

为了提高效率，减少磁盘IO，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存。这样做的理论依据是程序运行的局部性原理。

预读的长度一般为页（page）的整数倍。页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（在许多操作系统中，页的大小通常是4k），主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或者几页载入内存中，然后异常返回，程序继续运行。

B-/B+索引的性能分析

可以使用磁盘I/O次数评价索引结构的优劣。先从B-Tree分析，根据B-Tree的定义，可知检索一次最多需要访问h个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完成载入。为了达到这个目的，在实际实现B-Tree还需要使用如下技巧：

每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算存储分配都是按页对齐的，就实现了一个node只需要一次I/O。

B-Tree中一次检索最多需要h-1次I/O（根节点常驻内存），渐进复杂度为O(h)=O(logd(N))。一般实际应用中，出度d是非常大的数字，通常超过100，因此h非常小（通常不超过3）。

而红黑树这种结构，h明显要深得多。由于逻辑上很近的节点物理上可能很远，无法利用局部性，所以红黑树的I/O渐进复杂度也为O(h)，效率明显比B-Tree差很多。

所以，b-Tree作为索引的效率是很高的。

（6）连接的种类

外连接：包括左向外连接、右向外连接或完整外部连接。

左连接：left join或left outer join

左向外连接的结果集包括left outer子句中指定的左表的所有行，而不仅仅是连接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表均为空值（null）。

sql语句：select \* from table1 left join table2 on table1.id=table2.id

右连接：right join或right outer join

右向外连接是左向外连接的反向连接。将返回右表的所有行，如果右表的某行在左表中没有匹配行，则将为左表返回空值。

sql语句：select \* from t1 right join t2 on t1.id=t2.id

完整外部连接:full join 或full outer join

完整外部连接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。

sql语句：select \* from t1 full join t2 on t1.id=t2.id

返回左右连接的并集

内连接：内连接是用比较运算符比较要连接列的连接

内连接：join 或inner join

只返回符合条件的表的列

交叉连接（完全）：没有where子句的交叉连接将产生连接所涉及的表的笛卡尔积。第一个表的行数乘以第二个表的行数等于笛卡尔积结果集的大小。

交叉连接：cross join（不带where条件）

sql语句：select \* from t1 cross join t2

等价于select \* from t1,t2

（7）数据库范式

1）第一范式（1NF）（强调的是列的原子性，即列不能再分为其他几列）

在任何一个关系数据库中，第一范式是对关系模式的基本要求，不满足第一范式的数据库不是关系数据库。

所谓第一范式是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义实体，新的实体由重复的属性构成，新实体与原实体之间为一对多的关系。在第一范式中表的每一行只包含一个实例的信息。简而言之，第一范式就是无重复的列。

2）第二范式（2NF）（首先满足1NF，另外包含两个部分，一是表必须有主键，二是没有包含于主键的列必须完全依赖于主键，不能只依赖于主键的部分）

第二范式是在第一范式的基础上建立起来的，即满足第二范式必须先满足第一范式。第二范式要求数据库表中的每个实例或者行必须可以被唯一地区分。为实现区分通常要为表加上一个列，以存储各个实例的唯一标识。这个唯一属性列被称为主关键字或主键、主码。第二范式要求实体的属性完全依赖于主关键字，所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的唯一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

3）第三范式（3NF）（首先是2NF，另外非主键列必须直接依赖于主键，不能传递依赖）

满足第三范式必须先满足第二范式。简而言之，第三范式要求一个数据库表中不包含已在其他表中已包含的非主关键字信息。第三范式就是属性不再依赖于其他非主属性。（消除各表中的冗余）

（8）数据库优化思路

1）SQL语句优化

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。

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。

很多时候用exist代替in是一个好的选择。

用where子句替代having子句，因为having只会在检索出所有记录之后才对结果集进行过滤

2）索引优化

3）数据库结构优化

范式优化：比如消除冗余，节省空间；反范式优化：比如适当增加冗余，减少join；

拆分表：分区将数据在物理上分隔开，不同分区的数据可以制定保存在处于不同的磁盘上的数据文件里。这样，当对表进行查询时，只需要在表分区中进行扫描，而不必进行全表扫描，明显缩短了查询时间，另外处于不同磁盘的分区也将对这个表的数据传输分散在不同的磁盘I/O，一个精心设置的分区可以将数据传输对磁盘I/O竞争均匀地分散开。对数据量大的实时表采取此方法。可按月自动建表分区。

拆分又分为水平拆分和垂直拆分。

mysql可以容忍的数量级在百万静态数据可以到千万。

垂直拆分

解决的问题：表与表之间的io竞争，不能解决的问题：单表中数据量增长出现的压力

方案：把产品表放到一个server上，订单表放到一个server上。

水平拆分

解决的问题：单表中数据量增长出现的压力

不能解决的问题：表与表之间的io争夺

方案：用户表通过性别拆分为男用户表和女用户表

订单表通过已完成和完成中拆分为已完成订单和未完成订单。

4）服务器硬件优化

（9）存储过程与触发器的区别

触发器与存储过程非常相似，触发器也是SQL语句集，两者唯一的区别是触发器不能用execute语句调用，而是在用户执行transact-SQL语句时自动触发（激活）执行。触发器是在一个修改了指定表中的数据时执行的存储过程。通常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。触发器不同于存储过程，触发器主要是通过事件执行触发而被执行的，而存储过程可以通过存储过程名称名字而直接调用。当对某一表进行诸如update insert delete这些操作时，sql server就会自动执行触发器所定义的sql语句，从而确保对数据的处理必须符合这些sql语句所定义的规则。

=========================

存储过程是大型数据库中，一组为了完成特定功能的SQL语句集，存储在数据库中，在经过第一次编译后，调用不再需要再次编译，用户通过指定存储过程的名字并给出参数就可以执行它。

基本语法：

create procedure sp\_name

@[参数名][类型]，@[参数名][类型]

as

begin

……

end

=============================

触发器是一种特殊类型的存储过程。触发器主要是通过事件进行触发而被执行，而存储过程可以通过名字直接被调用。触发器的主要作用是能够实现由主键和外键所不能保证的复杂的参照完整性和数据一致性。

（10）sql优化

1）只返回需要的数据

a）不要写select \*语句

b）合理写where子句，不要写没有where的sql语句

2）尽量少做重复工作

可以合并一些语句

3）适当建立索引（不是越多越好）

建立索引后以下几点会进行全表扫描

a）左模糊查询’%...’

b）使用了不等操作符!=

c）or使用不当，or两边都必须有索引才行

d）in、not in

e）where子句对字段进行表达式操作

f）对于创建的复合索引（从最左边开始组合），查询条件用到的列必须从左边开始不能间隔。

g）全文索引：相当于文件建立一个以词库为目录的索引

4）使用join代替子查询

5）使用union代替手动创建的临时表

（12）存储引擎MyISAM、InnoDB

MyISAM、InnoDB区别

1）MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持

2）MyISAM表不支持外键，InnoDB支持

3）MyISAM锁的粒度是表级，而InnoDB是支持行级锁定。

4）MyISAM支持全文类型索引，而InnoDB不支持（5.6后支持全文索引）

5）MyISAM相对简单，所以在效率上优于InnoDB.

小型应用可以考虑使用MyISAM，当数据库有大量的写入、更新操作而查询比较少或者数据完整性要求比较高时选择InnoDB表。当数据库主要以查询为主，更新和写入比较少，并且业务方面数据完整性要求不那么严格，就选择MyISAM表。

一般情况下，在数据库中建立表的时候，每一张表强制添加 id 递增字段，这样更方便我们查询数据。

如果数据量很大，比如像订单这类，一般会推荐进行分库分表。这个时候 id 就不建议作为唯一标识了，而应该使用分布式的高并发唯一 id 生成器来生成，并在数据表中使用另外的字段来存储这个唯一标识。

首先使用范围查询定位 id （或者索引），然后再使用索引进行定位数据，即先 select id，然后在 select \*；这样查询的速度将会提升好几倍。