一）什么是存储过程？有哪些优缺点？

存储过程是一些预编译的SQL语句。

更加直白的理解：存储过程可以说是一个记录集，它是由一些T-SQL语句组成的代码块，这些T-SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。

> 存储过程是一个预编译的代码块，执行效率比较高
> 一个存储过程替代大量T_SQL语句 ，可以降低网络通信量，提高通信速率
> 可以一定程度上确保数据安全

如果你对存储过程不熟悉，建议阅读：存储过程详解-博客园

（二）索引是什么？有什么作用以及优缺点？

索引是对数据库表中一或多个列的值进行排序的结构，是帮助MySQL高效获取数据的数据结构

你也可以这样理解：索引就是加快检索表中数据的方法。数据库的索引类似于书籍的索引。在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库。

MySQL数据库几个基本的索引类型：普通索引、唯一索引、主键索引、全文索引

> 索引加快数据库的检索速度
> 索引降低了插入、删除、修改等维护任务的速度
> 唯一索引可以确保每一行数据的唯一性
> 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
> 索引需要占物理和数据空间

如果你对索引还不太熟悉，建议阅读：漫谈数据库索引

（三）什么是事务？

事务（Transaction）是并发控制的基本单位。所谓的事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性。

如果你对索引还不太熟悉，建议阅读：数据库事务 - Mr. David 专栏

（四）数据库的乐观锁和悲观锁是什么？

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和一致性以及数据库的统一性。

乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制采用的主要技术手段。

无论是悲观锁还是乐观锁，都是人们定义出来的概念，可以认为是一种思想。其实不仅仅是关系型数据库系统中有乐观锁和悲观锁的概念，像memcache、hibernate、tair等都有类似的概念。

针对不同的业务场景，应该选用不同的并发控制方式。所以，不要把乐观并发控制和悲观并发控制狭义的理解为仅在DBMS中存在的概念，更不要把他们和数据库中提供的锁机制（行锁、表锁、排他锁、共享锁）混为一谈。其实，在DBMS中，悲观锁正是利用数据库本身提供的锁机制来实现的。

悲观锁

在关系数据库管理系统里，悲观并发控制（又名“悲观锁”，Pessimistic Concurrency Control，缩写“PCC”）是一种并发控制的方法。它可以阻止一个事务以影响其他用户的方式来修改数据。如果一个事务执行的操作在某行数据上应用了锁，那只有当这个事务把锁释放，其他事务才能够执行与该锁冲突的操作。

悲观并发控制主要用于数据争用激烈的环境，以及发生并发冲突时使用锁保护数据的成本要低于回滚事务的成本的环境中。

悲观锁：正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守态度(悲观)，因此，在整个数据处理过程中，将数据处于锁定状态。 悲观锁的实现，往往依靠数据库提供的锁机制 （也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

悲观锁的流程：
1.在对某一记录进行修改前，先尝试为该记录加上排他锁（exclusive locking）。
2.如果加锁失败，说明该记录正在被修改，那么当前操作可能要等待或者抛出异常， 具体响应方式由开发者根据实际情况决定。
3.如果成功加锁，那么就可以对记录做修改，事务完成后就会解锁了。
4.其间如果有其他事务要对该记录做修改或加排他锁，都会等待该事务将该记录解锁或直接抛出异常。

MySQL InnoDB中使用悲观锁

注意：要使用悲观锁，必须先关闭mysql数据库的自动提交功能，因为MySQL默认使用autocommit模式，也就是说，当你执行一个更新操作后，MySQL会立刻将结果进行提交。

```
set autocommit=0;
```
- 1

```
//0.开始事务
begin;/begin work;/start transaction; (三者选一就可以)
//1.查询出商品信息
select status from t_goods where id=1 for update;
```

```
//2.根据商品信息生成订单
insert into t_orders (id,goods_id) values (null,1);
//3.修改商品status为2
update t_goods set status=2;
//4.提交事务
commit;/commit work;
```

上面的查询语句中，我们使用了select…for update的方式，这样就通过开启排他锁的方式实现了悲观锁。此时在t_goods表中，id为1的 那条数据就被我们锁定了，其它事务必须等本次事务提交之后才能对该记录进行操作。这样我们可以保证当前的数据不会被其它事务修改。

注意：上面提到，使用select…forupdate会把数据给锁住，不过我们需要注意一下锁的级别，MySQL InnoDB默认为行级锁。行级锁都是基于索引的，如果一条SQL语句没有用到索引是不会使用行级锁的，会使用表级锁把整张表锁住，这点需要注意。

优点与不足：
优点：悲观并发控制实际上是采用“先取锁再访问”的保守策略，为数据处理的安全性提供了保证；
缺点：在效率方面，处理加锁的机制会让数据库产生额外的开销，同时会增加产生死锁的机率；另外，在只读型事务中由于不会产生冲突，也没必要使用锁，这样做只会增加系统负载；还会降低并行性，一个事务如果锁定了某行数据，其他事务就必须等待该事务处理完才可以处理那行数

乐观锁

在关系数据库管理系统里，乐观并发控制（又名“乐观锁”，Optimistic Concurrency Control，缩写“OCC”）是一种并发控制的方法。它假设多用户并发的事务在处理数据时不会彼此互相影响，各事务能够在不产生锁的情况下处理各自影响的那部分数据。在提交数据更新之前，每个事务会先检查在该事务读取数据后，有没有其他事务对该数据做过修改。如果其他事务更新过该数据的话，正在提交的事务会进行回滚。乐观事务控制最早是由孔祥重（H.T.Kung）教授提出。

乐观锁（ Optimistic Locking ）是相对悲观锁而言，乐观锁假设数据一般情况下不会造成冲突，所以在事务对数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则返回错误信息，让用户决定如何去做。

相对于悲观锁，在对数据库进行处理的时候，乐观锁并不会使用数据库提供的锁机制，一般用记录数据版本的方式实现乐观锁。

数据版本：为数据增加的一个版本标识。当读取数据时，将版本标识的值一同读出，数据每更新一次，便对版本标识进行一次更新。当事务提交更新的时候，需要判断数据库表对应记录的当前版本信息与第一次取出来的版本标识是否一致，如果数据库表当前版本号与第一次取出来的版本标识值相等，则予以更新，否则认为是过期数据。

实现数据版本有两种方式，第一种是使用版本号，第二种是使用时间戳。

使用版本号实现乐观锁

使用版本号时，可以在数据初始化时指定一个版本号，每次对数据的更新操作都对版本号执行+1操作。并判断当前版本号是不是该数据的最新的版本号。

```
1.查询出商品信息
select (status,status,version) from t_goods where id=#{id}
2.根据商品信息生成订单
3.修改商品status为2
update t_goods
set status=2,version=version+1
where id=#{id} and version=#{version};
```

乐观并发控制假设事务之间的数据竞争(data race)概率比较小，因此尽可能直接做下去，直到提交的时候才去锁定，所以不会产生任何锁和死锁。但如果直接简单这么做，还是有可能会遇到不可预期的结果，例如两个事务都读取了数据库的某一行，经过修改以后写回数据库，这时就遇到了问题。

（五） 使用索引查询一定能提高查询的性能吗？为什么

通常,通过索引查询数据比全表扫描要快.但是我们也必须注意到它的代价.

索引需要空间来存储,也需要定期维护,每当有记录在表中增减或索引列被修改时,索引本身也会被修改. 这意味着每条记录的INSERT,DELETE,UPDATE将为此多付出4,5 次的磁盘I/O. 因

为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.使用索引查询不一定能提高查询性能,索引范围查询(INDEX RANGE SCAN)适用于两种情况:

基于一个范围的检索,一般查询返回结果集小于表中记录数的30%
基于非唯一性索引的检索

数据库事务的四个特征

ACID，分别对应Atomicity、Correspondence、Isolation、Durability

- 原子性：事务的所有操作，要么全部完成，要么全部不完成，不能停滞在中间环节。事务在执行过程中发生错误，会被回滚到事务开始前的状态，就像这个事务没有执行过一样。其保证了一致性，避免了事务执行当中（此时是不一致的）对外可见。
- 一致性：在事务的开始前和执行后，数据库的完整性约束没有被破坏。指事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。
    - 实际上数据库的一致性是人为定义的，该由编写该事务的程序员来确保。
    - 完整性检查：
        - 单个关系的完整性约束：not null、unique、check（谓词）等
        - 参照完整性，对应的操作有delete cascade、update cascade等
        - 断言：表示数据库任何时候都应该满足的一个条件
- 持久性：事务提交后，对系统的影响是永久的，即使出现系统故障。
- 隔离性：多个事务并发执行时，一个事务的执行不应影响其他事务的执行。虽然同一时间可能有多个事务会并发执行，但是系统保证，对于任意一事务来说，都感觉不到别的事务在并发执行，而是在它之前或者之后执行。
    - 脏读：A修改了数据，还未提交，B读取了这个修改的数据，最后A提交失败，那B读到的就是脏数据。（read uncommitted隔离会看到）
    - 不可重复读：一个事务多次读取同一数据，结果却不一样。第一次与第二次读取之间，数据被别的事务修改了。强调的是数据被修改了（read committed隔离会看到）
    - 幻读：和可重复读类似，但是另一个事务对数据的影响不是修改，而是新增或者删除，读取的记录数量前后不一致（repeated read隔离会看到，innodb默认的隔离级别为repeated read）
    - 常见的解决办法是串行执行，但是效率低。所以会有并发控制系统，来使事务并发执行，同时保证隔离性。（当我们将当前会话的隔离级别设置为serializable的时候，其他会话对该表的写操作将被挂起。可以看到，这是隔离级别中最严格的，但是这样做势必对性能造成影响。所以在实际的选用上，我们要根据当前具体的情况选用合适的。）

Read Uncommitted（读取未提交内容）：在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。

更详细的表述

Read Committed（读取提交内容）

这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。

由于正在读取的数据只获得了读取锁，读完之后就解锁，不管当前事务有没有结束，这样就容许其他事务修改本事务正在读取的数据。导致不可重复读。

解决不可重复读的问题就要求，对正在读取的若干行加上行级锁。要求在本次事务中不可修改这些行。解决ReadCommited更侧重数据行不可更新。

Repeatable Read（可重读）

这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读 （Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

解决幻读的方案应该是在表上加锁，幻读出现的场景主要是插入操作，由于插入操作使得事务不同的查询中出现不同的结果。

Serializable（可串行化） 这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

隔离级别越高，越能保证数据的完整性和一致性，但是对并发性能的影响也越大。对于多数应用程序，可以优先考虑把数据库系统的隔离级别设为Read Committed。它能够避免脏读取，而且具有较好的并发性能。尽管它会导致不可重复读、幻读和第二类丢失更新这些并发问题，在可能出现这类问题的个别场合，可以由应用程序采用悲观锁或乐观锁来控制。（乐观锁通过版本号控制是否存在不可重复读情况，如果不存在则提交，否则事务回滚。悲观锁是通过数据库系统本身在内部加锁，锁住要更新的数据，不允许其他事务修改，但是会消耗大量的性能）

SQL

group by 分组

1. 与聚合函数一同使用
2. 按多个字段分组
3. 与having一起使用，having相当于where，where不能与聚合函数联合使用
4. where group by having使用是有顺序的
5. 不支持group by all

order by

1. 接多个字段，靠前字段优先
2. 与group 联合使用时，order by 字段应出现在group by中或者聚合函数中

连接查询

1. 内连接
    1. 等值连接 与自然连接类似，全部选择 不删除重复列
    2. 不等连接 等号外的其他连接条件
    3. 自然连接 使用“=”，使用选择列表删除重复列 inner join on
2. 外连接
    1. 左外连接 选出左边列所有数据，当左边行数>右边行数时，右边不足行为null left join
    2. 右外连接 right join
    3. 全连接 full join mysql不支持全连接，可以使用union 合并左右连接实现
3. 交叉连接 不带where子句 cross join 显示所有组合，多余的均不显示

union 合并双方列数量，类型（类型可以不一样），顺序相同，消去重复行

union all 不消去重复行

distinct

1. 必须放在字段前面
2. 多个字段时，每个字段的不重复值会被列出

limit分页查询优化方法

1. 直接使用数据库提供的SQL语句 适用数据量小
2. 建立主键或唯一索引,利用索引
3. 使用预查询prepare
4. 子查询（利用id查出起始id，在此id上再进行查询），连接查询等
5. 如果id连续，可以使用id限定优化，即用between和limit结合使用

explain关键字

MySQL索引失效的几种情况

1. where查询语句中含有不等于号
2. where字句中的查询条件使用了函数
3. join操作中，MySQL只有在主键和外键数据类型相同时才使用索引，否则即使建立了索引也不会使用
4. where中查询条件使用了like 和regexp，nysql只会在搜索模板第一个字符不是通配符情况下才使用模板
5. 在order by操作中，mysql只有在排序条件不是一个查询条件表达式时才使用索引
6. 某数据列包含许多重复值，没必要建立索引
7. 使用or时，必须将or条件中的每列都带有索引才能有效
8. 如果列类型是字符串，一定要用引号将数据引用起来，即隐式转换
9. 如果mysql估计使用全表扫描要比使用索引快，则不使用索引

10. select *可能导致不走索引
11. not in ，条件满足下改成左连接方式可解决

in和exists的性能区别

1. in先进行子查询，即in后面的语句 exists先查主查询，即exists前面的语句
2. 如果子查询得出的结果集记录较少，主查询中的表较大且又有索引时应该用in,反之如果外层的主查询记录较少，子查询中的表大，又有索引时使用exists。
3. 其实我们区分in和exists主要是造成了驱动顺序的改变（这是性能变化的关键），如果是exists，那么以外层表为驱动表，先被访问，如果是IN，那么先执行子查询，所以我们会以驱动表的快速返回为目标，那么就会考虑到索引及结果集的关系了
4. in不对null数据进行处理

Innodb原理 https://www.cnblogs.com/shijingxiang/articles/4743324.html

1. 基于B+树
2. 表有多个字段，一颗B+树只能保存一个字段（索引），没有索引就变回了顺序查找
3. 为了解决多个B+树访问同一套表数据，使用聚簇索引（InnoDB）和非聚簇索引（MyISAM）
4. 聚簇索引是一种存储方式
5. 聚簇索引
    1. 聚簇索引存储来说，行数据和主键B+树存储在一起，辅助键B+树只存储辅助键和主键，主键和非主键B+树几乎是两种类型的树。
    2. 聚簇索引中行数据和叶子节点存储在一起，这样主键和行数据是一起被载入内存的，按照主键Id来组织数据，获得数据更快。
    3. 辅助索引使用主键作为"指针" 而不是使用地址值作为指针的好处是，减少了当出现行移动或者数据页分裂时辅助索引的维护工作，使用主键值当作指针会让辅助索引占用更多的空间，换来的好处是InnoDB在移动行时无须更新辅助索引中的这个"指针"。
6. 非聚簇索引存储来说，主键B+树在叶子节点存储指向真正数据行的指针，而非主键
7. Page结构
    1. 大小16K，有头部，内容，尾部构成
    2. Page连接起来是一个双向链表
    3. 有4种不同的Record（保存数据库一行数据），是一种单链表形式
        1. 主键索引树非叶节点
        2. 主键索引树叶子节点
        3. 辅助键索引树非叶节点
        4. 辅助键索引树叶子节点
8. 定位一个Record的过程
    1. 遍历B+树，通过各层非叶子节点最终到达一个Page，这个Page里存放的都是叶子节点（通过四种Record不同）
    2. 遍历Page内部Record，直到找到目标
    3. 未找到，遍历下一个Page

复合索引

1. 分窄索引（1-2列）和宽索引（2列以上）原则：用窄不用宽
2. 最左原则

数据库死锁原因

1. 两种锁类型
    1. 排他锁 其他的事务不能对它读取和修改。
    2. 共享锁 可以被其他事务读取，但不能修改。
2. 原因https://www.cnblogs.com/sivkun/p/7518540.html
    1. 两种资源c1,c2,A用户访问从c1(锁住)，B访问c2（锁住），此时A要访问c2,,B要访问c1，产生死锁

三大范式

①1NF：数据表中的每一列（每个字段）必须是不可拆分的最小单元，也就 是确保每一列的原子性；

②2NF：满足1NF后，要求表中的所有列，都必须依赖于主键，而不能有任何一列与主键没有关系，也就是说一个表只描述一件事情；

③3NF：必须先满足第二范式（2NF），要求：表中的每一列只与主键直接相关而不是间接相关，（表中的每一列只能依赖于主键）。

五大约束

1.primary KEY:设置主键约束；

2.UNIQUE：设置唯一性约束，不能有重复值；

3.DEFAULT 默认值约束；

4.NOT NULL：设置非空约束，该字段不能为空；

5.FOREIGN key :设置外键约束。

数据库索引优缺点

1.优点

（1）可以大大加快查询速度

（2）通过创建唯一性索引可以保证每一行数据的唯一性。

（3）可以加速表与表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

（4）在使用分组和排序子句检索时，可以减少分组和排序的时间

2.缺点

（1）创建和维护索引需要耗费一定时间，且随数据量增大而增大。

（2）索引需要占据一定空间

（3）当对表中数据进行增删时，索性也需要动态维护，降低了数据维护速度

.innodb与MyISAM的区别
①Innodb支持事务MyISAM不支持

②Innodb支持外键MyISAM不支持

③Innodb支持行级锁，MyISAM只支持表级锁

④InnoDB支持MVCC, 而MyISAM不支持